5/10/19
Saturday

classmate
Date
Page 1

LECTURE 1

- storing data of any application is must.
- we had file organi, for data storage.
  - Problem:
    - we were missing data abstraction.
      [hiding all internal details]

→ Any piece of data is stored in files only, but dbms gives flexibility to store as words/ tables.

↳ user whose data gets stored, but user can still fetch it : data abst provided in DB.

→ DBMS: facilitate the user to store, manage & access data from DB.

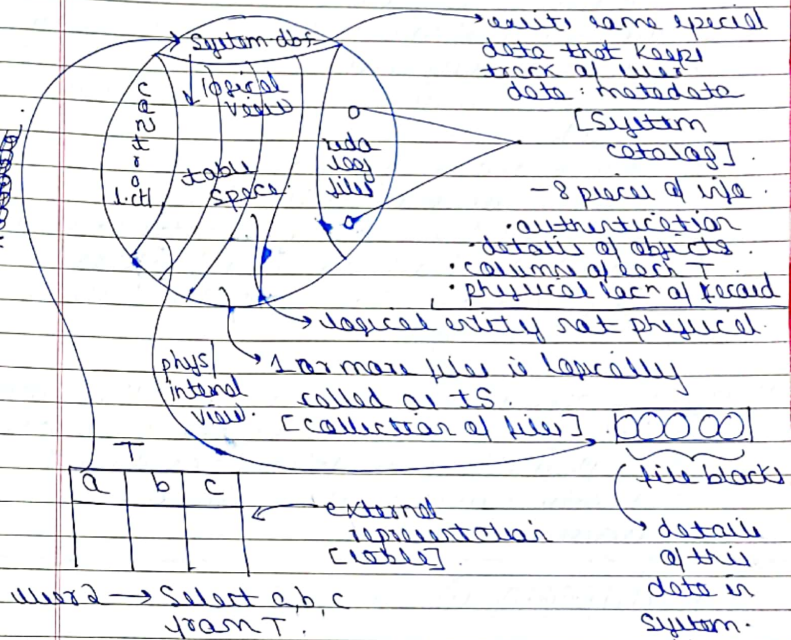
→ developed in 3 diff ways : [implemented]
- hierarchical DBMS
- network DBMS
- relational DBMS → most widely used.
  eg: Oracle, MySQL, SSMS

*	DBMS	RDBMS
	- had been a concept/ hypothesis.	implemented as RDBMS. It's a S/W.

- In file system, we know the exact folder & drive, we need full Physical path to access data, [data abs X]
- In RDMS, data abst get implemented in 3 views:
  (i) external view
  (ii) logical/conceptual
  (iii) physical/internal view.

* data: collection of related information.
* Collection of related data in an organised manner is called database.
* DBMS is a concept that facilitates the users while they store the data in the database, access the data from the database and manage the data of the database.
* DBMS provides the data abstraction feature.
* RDBMS is one of the implemented versions of DBMS which extends the data abstraction to 3 levels:
  (i) External / representational view.
  (ii) logical / conceptual view.
  (iii) Physical / internal view.

eg: Oracle Architecture [Oracle DB]



* As per Codd rules of rdbms, each and every rdb consists of some Special data that keeps track of all the details about the user data.
  Such Special data is called metadata.
* In every RDB, metadata will get stored in a separate location from the user data.

* Contents of Metadata:
1. The user credential information
2. All the list of objects that are owned by each and every user.
3. The list of columns in each and every object.
4. Logical location of the data.
5. Physical location of the data.
6. List of all the enforced integrity constraints
7. The details of all the created indexes.
8. The permission details of different users on different objects.

NOTE: user query (external view) hits the System dbs, gets converted to logical view & then to physical view, fetches the data & finally provides user with the representational view.

* INTEGRITY CONSTRAINTS

- These are the set of rules for maintaining the correctness of data in database. [integrity]
- There exists 2 types of integrity constraints:
(i) data integrity constraints.
(ii) referential integrity constraints.

(i) data integrity constraints

These constraints will validate the correctness of data with its value.
eg:
- unique constraint.
- Not NULL constraint
- check constraint.
- primary key constraint.

→ [NULL → neither 0 nor Blank].
↳ according to Codd's rule of RDBMS:
NULL V. ⇒ not assigned / not declared / not known value.

eg: unique (constraint defined on Col A), not NULL, check < 3L.

A	B	Salary
✓ 100	xyz	5L X
X100	xyz	
✓ NULL		

want to be able to check NULL/not

cant check whether unique/not

[Comb of unique & Not Null C→ Primary Key]

NOTE: 1. Unique constraint will be able to validate whether the value is unique or not, but it cant validate whether the value is null or not Null.

2. Not Null constraint can validate whether the value is null/not null; but it cannot validate whether it is unique or not.

* Functional dependencies → to understand keys.

- FD in RDB consists of determinant & dependent.
- The set of attributes/attributes that exist on the LHS of FD is called determinant.
- The set of attributes/attributes that exist on the RHS of FD is called dependent.
- As per the concept of FD, the value of the dependent gets determined by the determinant i.e. (determinant derives the value of dependent)

Determinant ⟶ (dependent)

eg: roll no ⟶ name (derives)

① If determ gets duplicated, value of dep also gets duplicated.

roll no	name
100	xyz
200	pqr
300	xyz
400	xyz

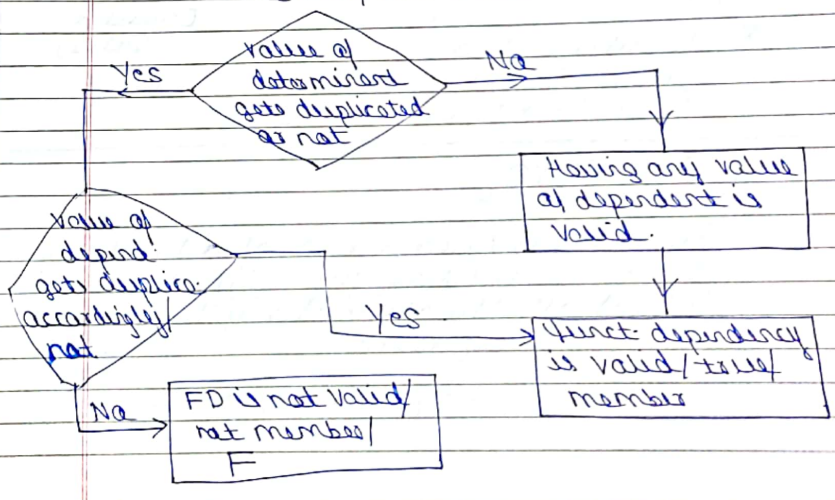
duplicated.

② In case det is unique, depend. can have any value.
⇒ Membership Test.

eg: Instance of relation is as follows:

A	B	C
1	2	3
2	2	3
3	3	2
4	2	3
5	3	2
6.	2	3.

Q. A → B T/F
Valid/not
member/not.



tD

$X \to X$

(1) Reflexive Axiom.
Each trivial dependency will always be Valid

* All possible FDs.

{A→A, A→B, A→C, B→A, B→B, B→C, C→A, C→B, C→C}

{AB→A, AB→B, AB→C, BC→A, BC→B, BC→C, AC→A, AC→B, AC→C}

{ABC→A, ABC→B, ABC→C}.

∵ A is unique, AB = unique ; AC = unique

(2) * decomposition axiom.
If A→BC is
⇒ A→B and A→C.

BC→BC is Valid
⇒ BC→B and BC→C [must be Valid].

(3) * Transitive axiom.
If A→B & B→C is valid
⇒ A→C, is also valid.

* SUPER KEYS

An attribute / combination of attributes of the relation is called Super Key if and only if all the attributes of the relation are derived by them.

{A, AB, AC, ABC} (derive all att)

relation ≡ Table
attribute ≡ column

1. SK can either be simple / composite.
   - simple: A
   - composite: {AB, AC, ABC}
2. A relation can have more than 1 Super Key.

* CANDIDATE KEY

An attribute / comb of attributes is defined as candidate Key if and only if:

(1) They derive all the attributes of the relation.

(2) They are the minimal Subset of Super Key.
(least of Subset (not based on no of attributes)).

ABC → {ϕ, A, B, C, AB, BC, AC, ABC}
(proper Subsets)
ABC → SK not CK.

ABC	CK X AC	CK X AB	CK ✓ A
ϕ	ϕ	ϕ	ϕ → in PS, none can derive.
A	A	A	A
B	C	B	
C	AC	AB	
AB			
AC			
BC			
ABC			

in PS of AB, A can derive all attrib
∴ this is further min level of Subsets].

(1) Candidate Key could either be Simple / composite.
(2) A relation can have more than 1 candidate Key.
(3) Minimal Subset is always validated on the level of Subset & not on the no. of attributes.

* NORMALIZATION → to reduce degree of redundancy.
NF ↑ dep of rel ↓

S1: find out the hipest NF.
S2: decompose table into higher NF.

* Redundancy

1)

Pname	FCP
X	Sachin
Y	
Z	Sachin
⋮	
	Sachin

→ repetition of data.

2)

Rollno	name	email	contact	course	due	fee
100	X	—	—	oracle	60^90	10K
200	Y	—	—	oracle	60^90	10K
300	Z	—	—	oracle	60^90	10K

IIA.

course → duration
→ fees.

redundancy cost many problems:
- wastage of memory
- updation anomaly
- insertion anomaly
- deletion anomaly

need to update in all rows.

- data redundancy would potentially cause the following problems ie

* NORMALIZATION

- It is the Systematic Process applied on the relations to reduce the degree of redundancy.
- Normalization Process always guarantees the following Properties:
  - lossless decomposition.
  - dependency preservation.

* NORMAL FORM

- It is the property of relation which indicates the amount of redundancy present in it.

- Normal form & degree of redundancy are inversely proportional
- When normalisation process gets applied on the relation, it performs the following steps of action:

(1) Find out the highest Normal Form of the given relation.
(2) Decompose the relation from its existing Normal form to higher Normal form.

** Procedure for finding highest NF of any Given relation:

Step 1: Find out all the possible candidate Keys of the given relation.

Step 2: Identify all the existing prime/Key attributes and Non-prime/non-Key attributes of the relation.

Step 3: Find out all the existing full dependencies, partial dependencies, transitive dependencies and the overlapping candidate key dependencies.

Step 4: Refer to the definitions and the hierarchy of Normal forms for finding the highest Normal form for given relation.

Q R(A, B, C, D)
{
A → B
B → C
C → D
D → A
}

Procedure to find out CK.

CK.
- (A)+ = {A, B, C, D} → either A, B, C alone or AB, AC, BC derive any new attributes, add it
- (B)+ = {B, C, D, A}
- (C+) = {C, D, A, B}
- (D+) = {D, A, B, C}

Q R(Sno, Pno, city, status, Qty)
{
Sno → city
Sno → status
city → status
Sno, Pno → Qty
}

- (Sno)+ = {Sno, city, status}
- (city)+ = {city, status} → decomposition

SK. CK (Sno, pno)+ = {Sno, pno, city, status, Qty}

∴ Sna, Pna cant derive all attributes.

* Prime/Key Attributes

The attributes of the relation that exist in atleast one of the possible candidate Keys of the relation are called Prime/Key attributes.

* Non-Prime/non-Key attributes

The attributes of the relation which doesnt exist in any of the possible candidate Keys are called non-prime/non-Key attributes.

NOTE: In order to identify whether an attribute of the relation is Prime/non-prime, we must find out all the possible CKs of that relation as the pre-requisite.

Q R(C,S,Z).

{ CS → Z ← check dependents. [if not CS/part of CS = only 1 CK
Z → C ← partial CS ∴ more than 1 CK possible
} ↑ replaced by can determin.

(CS)+ → {C, S, Z}
C → C ✗
S → S
∴ 2 CKs. ∴ {CS, ZS}

(ZS)+ → {Z, S, C}

Sno → City
Sna → State
City → State
SnoPna → Qty

∴ {Sna, Pna} are P/K attributes

P	P	NP/NK	NP/NK	NP/NK
R(Sna, Pna, City, State, Qty)				

P	P	P
R(C, S, Z)	∴ 0 non-Prime	

Q R(Sna, Pna, City, State, Qty)

(Unique & Not Null)

Sna	Pna	City	State	Qty
S1	P1	Bang.	100	1000
S1	P2	Bang.	100	1000
S1	P3	Bang.	100	50,000

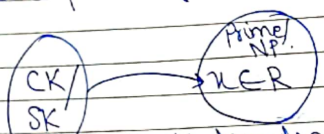
Insertion anomaly ⇒ S2 □ Pune 100 □ → cant insert part of raw

Deletion anomaly ⇒ S2 ~~P4~~ Pune 100 ~~10000~~ → cant also delete part of raw, ∵ CK becomes NULL.

Step3: FULL DEPENDENCY

$x \in R$

If an attribute x belongs to relation R, gets derived by either a CK or a SK then such dependency is defined as full dependency.



[full dependency].

- if the determinant is CK/SK, I dont care abt dependent, that dependency is FD.

* diagram representing FD:

NOTE: 1) While identifying the full dependency, we must validate the determinant of the functional dependency is either candidate Key/ Super Key.

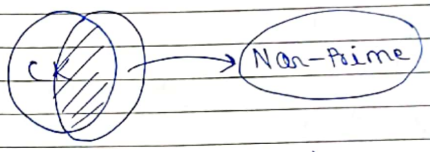
R(Sno, Pno, city, status, qty). (1NF)

{
Sno → city ✗ (PD)
Sno → status ✗ (PD)
city → status ✗ (TD)
Sno pno → qty ✓ (FD)
}

2) Full dependency will never cause redundancy and hence, normalization Process need not eliminate them.

* PARTIAL DEPENDENCY

If in case, a non-Prime/ non-Key attribute of the relation gets derived by only one part of the candidate Key, then such dependency is defined as partial dependency.



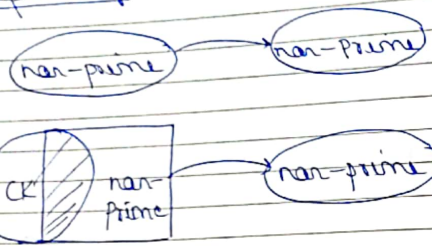
- 2 things to be noticed:

1) While identifying partial dependency, we must validate the determinant is part of the candidate Key & dependent is non-prime attribute.
2) Partial dependencies could cause redundancy in the relation hence Normalization process would try to eliminate them.
3) If there exists a relation whose property is having simple candidate Keys only, then such relations would never consist of partial dependencies.

* TRANSITIVE DEPENDENCY

- If a non-prime attribute of the relation gets derived by either another non-prime attribute / combination of part of the CK

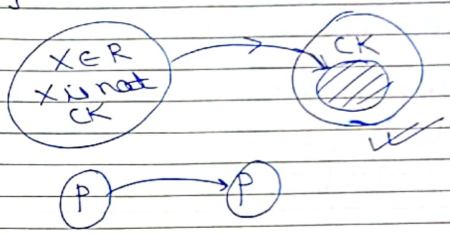
along with a non-prime attribute, then such dependency is defined as transitive dependency.



1. While identifying the transitive dependencies we must verify that dependent is a non-Prime attribute & the determinant is either a non-prime attribute / the comb. of Part of the CK along with a non-Prime attribute.
2. Transitive dependency could cause redundancy in the relations & hence normalisation process tries to eliminate them.
3. If there exists a relation which has only prime attributes then total possible partial & transitive dependencies in such relations are always 0.

* OVER LAPPING CANDIDATE KEY DEPENDENCY

R (C, S, Z)
{
CS → Z
Z → C
}

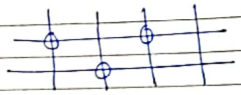


(1) If in case, atleast one of the possible candidate key of the relation is composite and if the part of the candidate key gets derived by another attribute combination. X ∈ R and X is not the candidate key (X is allowed to be part of the candidate key), such dependency would cause the CK to be overlapped.
Hence, it is defined as OCD.

Step 4

1] 1NF

- A relation is said to be in 1NF if and only if all the values in the relation are atomic.
- According to Codd's rule of RDBMS, each & every relation must be in 1NF minimum in the relational DB.
- no multivalued attributes allowed here.



2] 2NF

- A relation is said to be in 2NF if and only if:

(i) It exists in 1NF.
(ii) there are no Partial dependencies

Note: 2NF eliminates all the Partial dependencies & the redundancy caused by the Partial dependencies.

3] 3NF

- A relation is said to be in 3NF if & only if:

(i) It exists in 2NF.
(ii) It doesn't have any transitive dependencies

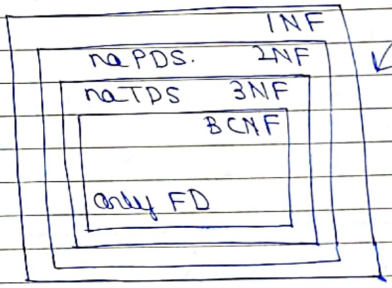
NOTE: 3NF eliminates both Partial & transitive dependencies hence, the redundancy caused by them.

4] BCNF

- Boyce-Codd Normal Form.
- A relation is said to be in BCNF if & only if:

(i) it exists in 3NF.
(ii) it doesn't have OCD.

Note: BCNF will allow only the full dependencies.



Hierarchy of Normal forms.

- No SQL DB → Big Data: → Volume, → Variety, → Velocity.
- structured data managed by RDBMS.
- → structured/semistructured [photos/videos].

- People switching to SQL DB → no concept of Normalization.

Q) Gate 2005

P P NP NP NP

R(F1, F2, F3, F4, F5)

{

F1 → F3 PD

F2 → F4 PD

F1F2 → F5 FD.

}

$(F_1)^+ = \{F_1, F_3\}$,

$(F_2)^+ = \{F_2, F_4\}$

$(F_1F_2)^+ = \{F_1, F_2, F_3, F_4, F_5\}$ FD

CK. INF

Q) R(Sno, Sname, Pno, qty)

P P P NP

{

Sno → Sname PD OCD

Sname → Sno PD OCD

Sno Pno → qty FD FD

}

2) Total no of existing Partial & transitive dependencies are.

(a) (1,1)

(b) (1,0)

(c) (0,1)

(d) (0,0)

(2) Highest NF of the relation is: ~~BNF~~ 3NF. (false)

$(Sno)^+ = \{Sno, Sname\}$

$(Sname)^+ = \{Sname, Sno\}$

$(Sno\ Pno)^+ = \{Sno, Pno, Sname, qty\}$

CK.

[(Sno, Pno), (Sname, Pno)]

(0,0)

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

{

AB → CDEF FD

F → G } TD

G → H

B → A FD. NO PD

C → D }

D → E } TD.

E → F

}

Non-prime.

1. Choose the correct statement wrt to candidate key.

(a) AB is the only possible CK.
(b) Along with AB there is are more CK
(c) Along with AB there are 2 more CK.
(d) None.

2. Highest NF of the relation is : 2NF

a) 1NF
b) 2NF
c) 3NF
d) BCNF

Find 1 & 2

① SK
$(AB)^+ = \{A, B, C, D, E, F, G, H\}$ ✓

$(F)^+ = \{F, G, H\}$

$(G)^+ = \{G, H\}$

② CK
$(B)^+ = \{B, A, C, D, E, F, G, H\}$ ✓

$(C)^+ = \{C, D, E, F, G, H\}$

$(D)^+ = \{D, E, F, G, H\}$

$(E)^+ = \{E, F, G, H\}$

closures

Q. R(A, B, C, D, E, F)
{
A → B PD
B → C
C → D
D → E
F → E PD.
}

(1) Choose the correct ans wrt to candidate Key:

a) there is no CK
b) insufficient data for finding CK.
c) A is the CK
d) None.

(2) Highest NF of the relation is : 1NF.

$(A)^+ = \{A, B, C, D, E\}$
$(B)^+ = \{B, C, D, E\}$
$(C)^+ = \{C, D, E\}$
$(D)^+ = \{D, E\}$
$(F)^+ = \{F, E\}$

(AF).

[AF]

Choose the best closure & combine the missing attribute.

→ Only 2 entities in DB → Strong & Weak.
→ Each & every design must have Primary Key.

PK	CK
- out of n, only 1 will be declared as 1 PK	- if n CKs

→ every relation must have atleast 1 CK

Q Gate 2004.

R(Name, rollno, coursename, grade)

{

(1) Name → rollno

(2) rollno → name

(3) name, coursename → grade FD

(4) rollno, coursename → grade FD

}

$(Name)^+ = \{Name, rollno\}$

$(rollno)^+ = \{rname, name\}$

CK: $(name, Cname)^+ = \{name, Cname, grade, rollno\}$

CK: $(rollno, Cname)^+ = \{rname, Cname, grade, name\}$

3NF ✓

→ No PD, No TD.

R(N, R, C, T, Z)

{

N → RCT FD

RCT → Z TD

Z → CT TD.

}

2NF ✓

$(N)^+ \rightarrow \{R, C, T, Z, N\}$

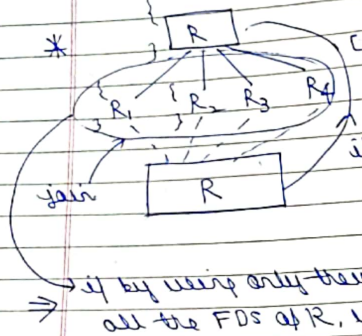
$(RCT)^+ \rightarrow \{R, C, T, Z\}$

$(Z^+) \rightarrow \{Z, C, T\}$

N → CK

6/10/19 Sunday

Lecture 2



⇒ if by using only these FDs, you can derive all the FDs of R, we could confirm property: dependency preservation [guarantees preserval of all functionality all integrity constraints].

* Lossless decomposition

There exists relation R. that got decomposed into set of relations: $R_1, R_2, R_3 \ldots$ By applying join on these relations if we could derive all the data as in the original relation, the property is defined as lossless decomposition.

* Dependency Preservation

Relation R gets decomposed into set of relations $R_1, R_2, R_3 \ldots$ by making use of all the projected functional dependencies if we

could derive all the FDs of the given relation. Then it is dependency Preservation.

NOTE: 1. lossless decomposition guarantees the availability of all the data & dependency Preservation guarantees the availability of all the integrity constraints.

* Step 2 of normalization

R(Sno, Pno, City, Status, qty)
P P NP NP NP

{
Sno → City PD
Sno → Status PD
City → Status TD
Sno Pno → qty FD
}

1NF

decompose into 2NF.

R

R_1 2NF R_2

Sno City Status | Sno Pno Qty

{ FD Sno → City
FD Sno → Status
TD City → Status
} 2NF

{ Sno Pno → Qty FD. }

BCNF → 2NF.

decom into 3NF.

R_1

$R_{1,1}$ BCNF ⇒ 3NF. $R_{1,2}$

City Status | City Sno

{ City → Status FD } | { Sno → City } FD.

∴ BCNF ⇒ 3NF

D(city Sno/ city status/ SnoPno Qty)

↳ decomposition lossless/ not?

2 Step Algo

* Procedure for finding whether the given decomposition is lossless or not.

Step 1: Create the instance of the given relation by taking the attributes of the relation as columns & the given decomposition as rows.
Fill the values of the instance with X & Y.
- X value if the attribute is present in the decomposition.
- else, place the Y value along with row & column nos.

Step 2: Make sure all the FDs of the given relation are valid for the instance we prepared. During the validation, we could change:

$Y \longrightarrow Y$

$Y \longrightarrow X$

but, $X \longrightarrow Y$ (X). [Wrong]

Step 3: Either during the validation of FDs or after the validation of FDs, check if there exists atleast one row with all the X values.

- If such row does exist the given decomposition is lossless decomposition.

→

	Sno	Pno	city	status	Qty
Sno city	X	Y_{12}	X	~~Y_{14}~~	~~Y_{34}~~ Y_{15}
city status	Y_{21}	Y_{22}	X	X	Y_{25}
SnoPno Qty	X	X	~~Y_{33}~~ X	~~Y_{34}~~ X	X

↳ all X in row ∴ lossless

check the FDs.
Sno → city ✓
Sno → Status ✓
city → Status ✓ [no need to check further].

* Procedure for Validating the dependency Preservation Property.

Step 1: Generate all the possible FDs by finding the closures of determinants of FDs of the given relation.

Step 2: Apply the projection on all the applicable FDs for finding the members of each decomposition.

Step 3: Make use of only such projected FDs and check whether all the FDs of the original

relation could get derived or not.
If they gets derived, dependencies are preserved.

$(Sno)^+ = \{Sno, city, status\}$ —— Sno → Sno
Sno → City
Sno → status X

$(city)^+ =$ —— city → city
city → status.

$(Sno\,Pno)^+ =$ —— Sno Pno → Sno
Sno Pno → Pno
Sno Pno → city X
Sno Pno → status X
Sno Pno → Qty.

from decomp

{ city, Sno } { city status } { Sno Pno Qty }
{ Sno → Sno, Sno → city, city → city } { city → city, city → status } { Sno → Sno, Sno Pno → Sno, Sno Pno → Pno, Sno Pno → Qty }

using these FDS, can you derive the FDS of eg relation.

Sno, Pno, City, S, Q)

Sno → City ✓
Sno → status ✓ [Trans]
City → status ✓
Sno Pno → Qty ✓

∴ dependency Preservation ✓

* BCNF ALGORITHM

- These algorithm can be used for decomposing any given relation directly into BCNF
- These also gives guarentee for:
  1. The final BCNF decomposition.
  2. The lossless decomposition.

These algo fails to give guarantee for dependency Preservation also. (sometimes it will, I am. it want)

- it will identify the dependencies, which are not full dependencies.

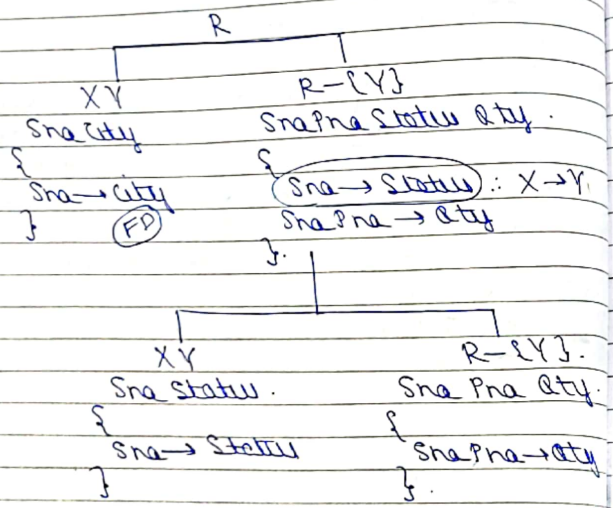
Step1: Identify the functional dependencies that are not full dependencies
Consider them as $X \to Y$.

Step2: Start decomposing the relation R using any of such $X \to Y$ into XY and $R - \{Y\}$.

Step3: Identify both the decomposition are in BCNF or not.
If not reapply the algorithm on the decomposition which is not in BCNF.

R(Sno, Pno, City, Status, Qty)

{
Sno → City X→Y PD
Sno → Status PD
City → Status TD.
Sno Pno → Qty FD.
}

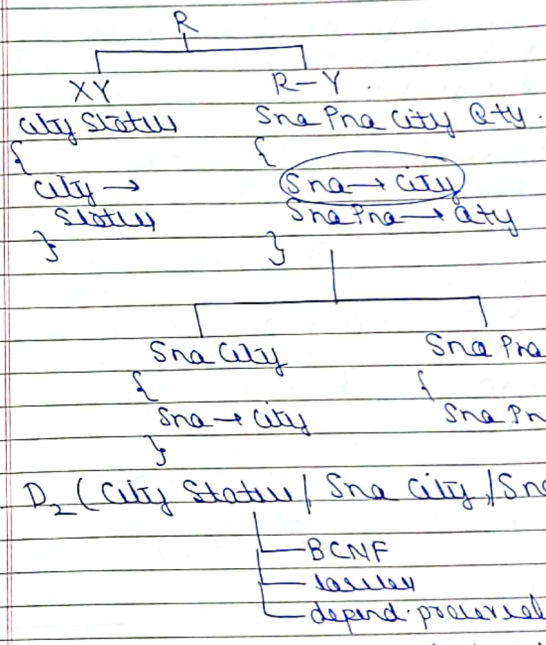


∴ D_1 (Sno City | Sno Status | Sno Pno Qty)

3 Relations in BCNF.

- BCNF
- lossless
- dep not preserved ∵ City → Status not preserved

But, if perform alpha on City → Status (X → Y)



∴ D_2 (City Status | Sno City | Sno Pno Qty)

- BCNF
- lossless
- depend. preserved.

[Sno → Status derived due to transitive AXIOM]

NOTE: Though BCNF gives guarantee for zero redundancy, it is not the preferred normal form practically as it is uncertain during the dependency preservation.

COVER

- There exists 2 functional dependency sets F and G. We could define G covers F if and only if all the FDs of F are implied by G.
- If in case, F covers G & G covers F, then we could define F & G are equivalent.

F	G
~~A→B~~	
~~A→C~~	A→BC
A→B	A→B
B→C	B→C
~~A→C~~ (Can be derived by trans.)	A→C

G
A→B
B→C

∴ F ⊆ G & G ⊆ F.

G covers F.

⇒ F ≃ G

3NF DECOMPOSITION ALGO

- This algo is used for decomposing any given relation to 3NF directly.
- This algo gives guarantee for:
  (i) the final 3NF decomposition.
  (ii) dependency preserving decomposition (due to cover).
  (iii) lossless decomposition.

Step 1: Find out the cover of functional dependency set of the given relation.

Step 2: Decompose the relation according to the cover we got in Step (1)

Step 3: Verify if the decomposition is lossless or not: if not add the candidate key of the given relation as a seperate decomposition.

There exists 2 types of covers:
(ii) Minimal cover
(iii) Canonical cover

Processes for finding Minimal cover:

Step 1: Apply the decomposition axiom on all the functional dependencies to ensure that dependent of each FD has only 1 attribute.

Step 2: Eliminate the unnecessary attributes that exist on determinant side of functional dependencies.

Scenarios for applying Step 2:

AB→C, B→C	B→C.
~~A~~B→C, A→C	B·A→C: A
AB→C	BA→C and
A→B	BA→A.

~~A~~B→C.
B→C.
~~A~~B→C
B→A.

Step3: eliminate the unnecessary FD which could be either redundant FD / transitive FDs.

* Procedure for finding canonical cover

Step 1: Combine the FDs whose determinant is same.

Step 2: Eliminate the unnecessary attributes that exist on ~~dependent~~ determ side of FD.

Scenarios discussed during the minimal cover are applicable in canonical cover procedure also.

Step 3: Eliminate the unnecessary attributes that exist on dependent side of FD.

Scenarios for Applying S3

AB → C
AB → D
C → D

AB → C ✗
C → D
✓

AB → D
A → C

AB → C
AB → D
AB → C

AB → CB
AB → C & AB → B

AB → C ✗
BC → D

$(AB)^+ = \{A, B, C, D\}$
$(AB)^+ = \{A, B, C, D\}$

$(BC)^+ = \{B, C, D\}$
$(AB)^+ = \{A, B, C, D\}$

Step 4: eliminate the unnecessary FDs

P P NP NP NP

Q. R(A, B, C, D, E)

AB → CD	FD
D → E	TD
A → C	PD
B → D	PD.

→ 1 NF

CK: $(AB)^+ = \{A, B, C, D, E\}$
$(D)^+ = \{D, E\}$
$(A)^+ = \{C, A\}$
$(B)^+ = \{B, D\}$

decompose into 3 NF.

S1): find cover. (either min/can).

AB → C	AB → C	A → C
AB → D	AB → D	B → D
D → E	D → E	D → E.
A → C	A → C	A → C
B → D	B → D	B → D

Minimal Cover

AC BD DE AB

add CK. dependency preserving.

* RELATIONAL ALGEBRA

- Collection of methods that are applied for retrieving data as per user requirement is called relational Algebra.
- The methods we apply are the operators of relational Algebra.
- Such operators are divided into 2 types:
  (i) Set theory / theoretic → ∪ ∩ −
  (ii) Native operators.
    - σ Selection
    - π Projection
    - ρ rename
    - ÷ divide
    - ⋈ Join.
    - ×

* Degree

- Total no. of attributes present in a relation or total no. of columns present in a Table is called degree of the relation.
- degree of relation R is represented as d(R).

* Cardinality

- Total no. of rows present in a table or the tuples present in a relation is called cardinality of the relation.

|R| ← cardinality.

* Domain

Total range of accepted values for an attribute of the relation is called domain of the attribute.

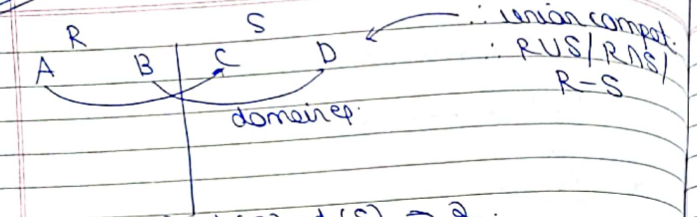
eg: create table T
{
  a char → range of accepted values.
  b int
}

* Union Compatibility

Two relations R and S are said to be union compatible with each other if and only if:
(1) they have the same degree.
(2) The domains of the respective attributes is same in both the relations.

- In order to apply union, intersection & minus operators among the relations, they must be union compatible with each other.

Set Theoric



: union compat.
: $R \cup S / R \cap S / R - S$

- $d(R \cup S) = d(R) = d(S) \Rightarrow 2$.
- $d(R - S) = d(R) - d(S)$

↳ present in R, not present in S.

1). $d(R \cup S) = d(R) = d(S)$
$\max(|R|, |S|) \leq |R \cup S| \leq |R| + |S|$

2). $d(R \cap S) = d(R) = d(S)$
$0 \leq |R \cap S| \leq \min(|R|, |S|)$

3). $d(R - S) = d(R) = d(S)$
$0 \leq |R - S| \leq |R|$.

(R1)

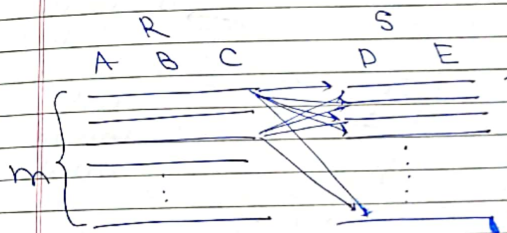
- ~~Intersect~~

* CROSS PRODUCT (CARTESIAN)

$$d(R \times S) = d(R) + d(S)$$
$$|R \times S| = |R| * |S|$$

- Relations need not be union comp for CP.

⇒ If in case, the application/user query requires the type of data which can't be extracted from any of the tables directly.
⇒ The data extraction requires the association of columns of relation R with the columns of relation S.
⇒ In order to apply Cartesian Product on any 2 relations, they need not be union compatible with each other.
⇒ Cartesian Product Produces all the possible associations among the rows of the given relations in the result set.



∴ $m * n$ rows in ~~MATRIX~~ R X S

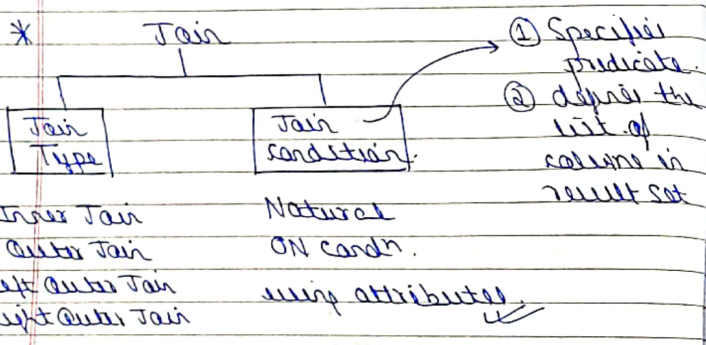
R.A	R.B	R.C	S.D	S.E

mn.

* JOIN

- Result of the Join operator is the subset of the result of the cartesian Product
  Hence, operator 'Join' is considered as the subset of the operator 'Cartesian Product'.
- Join operator has 2 sections:

1) Join condition.
2) Join Type



* Join Condition

It is primarily responsible for:
(i) Specifying the predicate used for Join.
(ii) Defining the list of attributes that would be the part of the result set.

→ There exists 3 Join Conditions:

(i) Natural condition

(1) In order to apply this condition on any 2 relations, there must have atleast one common column.
(2) This generates the predicate on the common columns with equal to comparison operator only.
Hence, also called "EQUI JOIN"
(3) This join condition eliminates the duplicates of the common columns in the result Set.

(ii) ON Condition

(1) We can apply this join condition on any of the two relations irrespective of they have common column / not.
(2) Within the join Predicate, we could make use of any of the comparison operator [THETA JOIN]
(3) This join condition will not eliminate any of the columns in the result Set.

(iii) USING Attributes Join

(1) This is the special Case of Natural Join in which some/all the common columns can be used in the join predicate.
(2) The duplicates of the columns that are used in the predicate gets eliminated in the result Set.
(3) If all the common columns are used, then result of this Join condition will exactly be same as the result of Natural Join.

* JOIN TYPE

This component of Join primarily responsible for:
(1) Finding whether the rows are matched or not as per join condition.
(2) Decision on including/excluding the unmatched rows in the result Set.

(i) INNER JOIN

This Join type will allow only the matched rows in the result Set.

(ii) Left Outer Join

This Join Type allows the following:
(1) The Matched rows.
(2) The unmatched rows of left Side relation along with Null Values for the columns of right side relation.
(3) Left Outer Join rep deficiency on the right Side relation.

(iii) Right Outer Join

This Join type will allow:
(1) The Matched rows.
(2) UnMatched rows of right Side relation along with Null Values for the columns of left Side relation.
(3) ROJ represents deficiency on the Left Side relation.

(iv) Full Outer / Outer Join

This Join type will allow the matched rows, unmatched rows of both left & right Side relations along with Null Values for the respective positions in the result Set.

> Left Outer Join ∪ Right Outer Join
> = Outer Join

R

A	B	C
1	b_1	c_1
2	b_2	c_2
3	b_3	c_3

S

A	B	E
1	b_1	e_1
3	b_2	e_2
4	b_3	e_3

R.A = S.A
&
R.B = S.B
} Natural condition

Type of Join.

A	B	C	E	
1	b_1	c_1	e_1	← Inner J.
2	b_2	c_2	NULL	← Left outer J.
3	b_3	c_3	NULL	
3	b_2	NULL	e_2	right outer J
4	b_3	NULL	e_3	

↳ Full outer Join.

NOTE: Join operator consider a row as qualified if it gets matched atleast once.
[like 1 b_1 c_1]
• To represent all "atleast once" scenarios, join operator should be applied.

* DIVIDE

- In order to apply R ÷ S, the pre-requisite is the list of attributes of the right side relation should always be the subset of the list of attributes of left side relation.
- We would get the values of attributes of left side relation in the result that got associated with ALL the values of right side relation.

Note: Divide considers a row as qualified if & only if, it gets matched with all the required values.

R ÷ S

S ⊆ R

R

Sno	Pno
S_1	P_1
S_2	P_1
S_1	P_2
S_2	P_2
S_3	P_1
S_4	P_2
S_5	P_1
S_6	P_2

S

Pno
P_1
P_2

Sno
S1
S2

* Selection [in Relational Algebra]

- This operator of relational Algebra would produce the logical, horizontal subset of rows that are qualified by the predicate.
- The result of the Selection operator in RA would be same as the result of where clause in SQL.

$$d(\sigma_{Result}) = d(original)$$

$$|\sigma_{result}| \leq |original|$$

eg: $\sigma_{deptno=10}$

* Projection

- This operator of RA would produce the logical, vertical subset of rows by eliminating the duplicate values in the result set.
- This operator is equivalent to "Select with distinct" clause of SQL.

$$d(\Pi_{result}) \leq d(original)$$

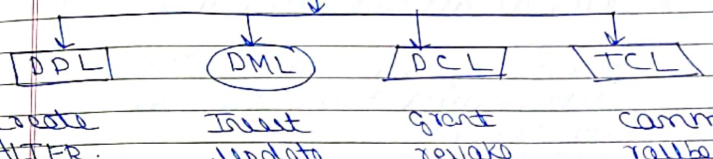
$$\because |\Pi_{result}| \leq |original|$$

$\because$ gets rid of duplicate values

* Rename

- This operator of RA would neither change the degree nor the cardinality.

* SQL → Ansii Standard SQL.



DDL	DML	DCL	TCL
create	Insert	grant	commit
ALTER.	Update	revoke	rollback
RENAME	delete		
TRUNCATE	Select → 99% Q in Gate.		
DROP			

1] All the DDL commands would exhibit the auto commit Property.
2] When any of the DDL commands get executed, the metadata gets updated automatically.
3] Every relational data object would consist of 2 sections of information:
   (i) Metadata of the object.
   (ii) Userdata of the object.
4] When truncate command is issued, only user data gets purged, & metadata remains as it is.

5] Whenever drop commands executor will purge not only the user data but also metadata.

* Truncate v/s delete

- Truncate command is used for purging all the rows of the table at a time.
- Delete can be used for purging either one or some or all the rows of the table.

Truncate requires only one step.
Delete requires two steps for the execution.

* Select

M	⑥ Select.	
M	① From	<Source of data>
rows O	② where	<predicate>.
X O	③ Group by.	<expression>.
groups O	④ Having	
O	⑤ order by	

Q on Where

Special opr in Query

9/1

Q. Select *
from emp
where deptno = 10
or
deptno = 20
and
sal > 5000

exp1 inside exp2.

OR
AND
NOT

9/2

Select *
from emp
where (deptno = 10
or
deptno = 20)
and
sal > 5000

(deptno = 10 or deptno = 20) = Exp1
Exp1 and sal > 5000 = Exp2.

9/2 ≤ 9/1

* where deptno = 10
or
deptno = 20
or
deptno = 30

≡ where deptno IN (10, 20, 30)

Note: If all the list of values of comparision are provided, we could make use of a special operator or 'IN' within the the predicate.

in Between part lower limit first.

* Select *
from emp
where sal ≤ 5000
and
sal ≥ 2500

equal

Select *
from emp
where Sal BETWEEN
2500 and 5000.

① If the range of values of comparison is provided, then we could use between operator in the Predicate.
② While between gets used, the following points must be valid.
(i) Always the limits of comparison should be included in the predicate.
(ii) Between exp must consist of lower limit first followed by the higher limit.

* LIKE PREDICATE

- Like Predicate gets used for performing the String matching in SQL.
- There exists 2 special char that are used within the like Bracket: _ & %
- 'underscore' can be used while matching one char at a time.
- '%' can be used for matching 0/more char at a time.

where Ename like '_ _ _ _' ← 4 char name.
like 'S%'.

* Select *
from emp
where Job like 'SA#_%'

starts with SA_
escape '#'

Job
Salesman
Salesmanager
SA_Rep
SA_Boy
SA_girl

* FUNCTIONS IN SQL

→ atleast 1 value must be returned to calling

→ Unlike the normal Prog. language functions, in SQL, each & every functn must & should return atleast 1 value to the calling environment.
→ In SQL, functions divided into 2 Types:

1. Single Row function
2. Multiple row / Group / Set / Aggregate functions.

- These functions could be applied on 1 row at a time to get one result per row
eg: case translation functions
Upper / Lower / Initcap.

② Character functions.

eg: Concat, Stolen, Instring, Substring

③ Number functions

eg: Seal / Floor etc.

④ Date functions.

eg:

MONTHSBETWEEN, GETDATE() etc.

⑤ Conversion functions

- These functions can be applied for converting char to date datatype / vice-versa.

* Multiple Row Function

- These functions can be applied on multiple rows of the table at a time to get 1 result.
- MR could either be:
  - all the rows of the table
  - the rows of the table splitted into groups using group by clause.

eg: max(), min(), sum(), avg(), count()

where → one row at a time.

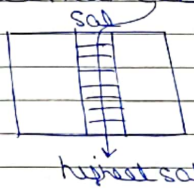
Q. Select empno, deptno, Sal
from emp
where sal = max(Sal).

↳ not compatible

- Only Single row functions are compatible to be used in the where clause.
- Group functions are not compatible to be used in the where clause.

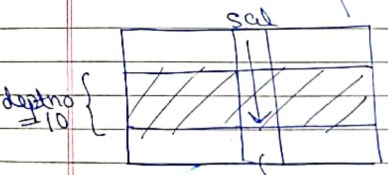
* Group By

1) Select max(Sal) from emp



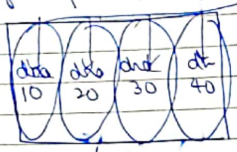
↳ highest sal.

2) Select max(Sal) from emp
where deptno = 10



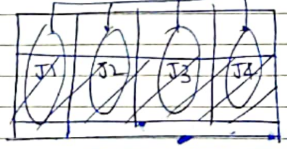
↳ highest Sal of deptno 10.

3) Select max(Sal)
from emp
group by deptno.



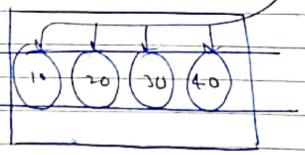
↳ return 4 values (max sal of each department).

4) Select max(Sal)
from emp
where deptno = 10
group by job.



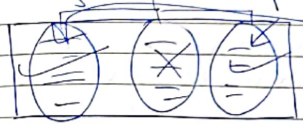
dept no = 10

5) Select count(empno), deptno
from emp
where sal > 1L
group by deptno.



Gate

6) Select sum(Sal), deptno
from emp
group by deptno
Having count(empno) > 4



7) Select max(Sal),
from emp
group by deptno

5000
8000
6000

deptno can be P/A.

grp exp

8) Select max(Sal), deptno
from emp
group by deptno;

✗ error.

non-group attributes present.

NOTE: All the non-group expressions that exists in the Select clause, along with group expressions must & should be present in the group by clause else, the query returns an error.

- It is not vice-versa.

* Order By

- By default, SQL displays the result set in the same order as the data got inserted.
- In case if we want to override that behaviour, we can make use of **order By clause**.
- If multiple expressions are used within the order by clause, then Sorting takes place based on exp 1 first, on top of it's result exp 2 gets applied and so on.

eg:

```
Select deptno, sal
from emp
order by deptno, sal desc;
```

10	5K ↑
10	4K
10	3K
20	8K ↑
20	7K
20	6K
30	9K ↑
30	8K

```
Select deptno,
sal
from emp
order by
①, ② desc
```

↓ same output.

→ We could also specify the column no in the order by expression based on the Select query.

* SubQueries

- If in case, the Predicate based on which the query has to be computed is not ready for the execution then we would end up writing 1 query inside another query.
- The inner query is also called SubQuery & the outer query is also called Main Query.
- Always, the SubQuery will complete the execution first. After the completion, it could return either 1 value/ many values to the main Query.
- If the SubQuery returns only 1 value, it is called, Single row SubQuery
- If the SQ returns multiple values, it is called, multiple row SQ.

→ In case of MR SubQ: we should make use multiple row operators for making the comparison compatible.

eg:

```
Select *
from emp
where sal = ( Select max(Sal)
              from emp )
```

1 value - Single row.

① * Select max (Sal)
from emp
where sal < (Select max (Sal)
from Emp ← 15000
where
Sal < (Select max(Sal)
from emp))

Sal
10K
5K
8K
12K ×
15K ×
9K

→ 3rd highest salary. [10K]

* Multi Row SQ

Select emp no, Ename, deptno, Sal
from emp ← return multirow
where deptno IN (Select deptno
from dept
where location =
'Pune')
and
sal > 50000

Goto 2001

if '=' → error.
∴ IN ✓

Q Select E.empno, E.ename,
from Emp E, Dept D
where E.deptno = D.deptno
and
D.location = 'pune'
and
E.Sal > 50,000. ← JOIN
same d.p as 2001.

Q Select *
from Emp
where Sal > ALL (Select Sal
from Emp
where Job = 'CLERK'
)

≡

where sal > (Select max (Sal)
from emp
where job = 'CLERK')

→ details of the employees whose salary is greater than the max salary of clerks

- In General, the normal Subqueries gets executed only once irrespective of no. of rows we have in the Table.

* CORRELATED SUBQUERIES

- These are the special type of SubQueries which gets executed no. of times equal to the no of rows we have in the table.
- We could identify the correlated SubQuery by making use of the column belongs to the outer Query, inside the inner Query.

* Execution Procedure

Step1: Main Query will identify the rows for processing rows that is candidate row.

Step2: Main Query will pass the candidate row values to the SubQuery.

Step3: SubQuery will complete the execution once based on the candidate row values it receive & returns the result to the main Query.

Step4: Main Query takes the decision of either including/excluding that row in the result Set before it finds the next candidate row value.

Step5: The above procedure gets repeated for all the candidate row values.

eg: from Emp A ← alias of emp relation

Q Select A.sal
from Emp A
where 4 = (Select count(distinct(B.Sal))
from emp B
where B.sal > A.sal).

4th rank. (q-1)
nth rank. (n-1)

5th rank

A emp Sal	B emp sal
✗ 12K	12K
✗ 36K	36K
✓ 10K	10K
8K	8K
15K	15K
18K	18K
12K	12K

3 6 15 12 10

* In SQL, we can use Set operators for combining the results of 1 or more Select queries.
- When Set operator is used, the Query collectively is called Compound Query.
- The individual Select statements are referred as component Queries.
- The component queries must be union compatible with each other.

$\{J_1, J_2, J_3\} \cup \{J_2, J_4, J_6\}$
$\{J_1, J_2, J_3, J_4, J_6\}$

* Select Job
from Emp
where deptno = 10.
UNION
Select Job
from Emp
where deptno = 20

→ will eliminate duplicates.

Select Job
from emp.
where deptno = 10
and
deptno 20

→ won't eliminate duplicates.
[add select dult. job]

* $\{J_1, J_2, J_3\} \cap \{J_1, J_2\}$
$\{J_1, J_2\}$

* Select Job
from Emp
where deptno = 10
INTERSECT
Select job
from Emp
where deptno = 20

→ Jobs where dept no is 10 and 20 both.

Select job
from emp
where deptno = 10
and
deptno = 20

→ no result.
∵ where neptune row
∴ [not possible]

H/W
1) diff b/w commit & checkPoint.
2) What is NP complete Problem.
3) Describe Shadow Paging.
4) What is WAL Protocol
5)

12/10/19
Saturday

Date Page 58

Lecture 3

* TRANSACTION MANAGEMENT → min 1MQ.

consistent state = data must be correct.

⇒ A sequence of instruction which could convert the dataBase from 1 consistent state to another consistent state is called transaction.

⇒ A group of DML instructions which could convert the database to a consistent state by executing all the instructions atomically is called transaction.

⇒ Each & every such transaction in RDBMS must satisfy the following properties:

(i) Atomicity
(ii) Consistency
(iii) Isolation
(iv) Durability

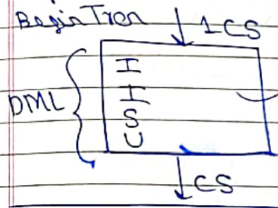
} ACID ✓

1) Atomicity

- According to this property of transaction, either all instructions of transaction should get executed or none of the inst. of transaction should get executed.
- In oracle RDBMS, the inst. of the transaction

gets embedded between:
begin tran & end tran commands
- Among them, Specifying begin tran is optional and end tran is mandatory.

NOTE: End of 1 transaction indicates the begin for the next transaction. ✓



2) Consistency

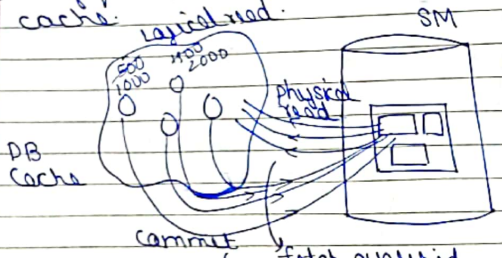
- Accord. to this property of transactions, data in the data Base must remain consistent not only before the transaction but also, after the transaction. irrespective of the transaction gets commited / roll
- RDBMS will make use of the concept called Shadow Paging for guaranteeing the consistency Property.

* Shadow Paging

According to this Property RDBMS keeps track of the following details:

1) Copy of the data item before the transaction.
2) Copy of the dT after the transaction
3) Details of the transaction that performed the modification.

- only qualified rows are fetched into PB cache.



> Update emp
Set Sal = Sal + 500
where deptno = 20
> commit

RDBMS guarantees after Trans fails write back to disk.

Rollback (abort): the modification are ignored and must get replaced with the before transaction values.

* End of the transaction could either be commit / rollback.
In case of COMMIT:

all the modifications performed by the specific transaction are assured to be transferred from the volatile cache to the non-volatile disk.
In case of ROLLBACK:

The abort operation takes place. Abort operation has the following steps:
(i) Ignore all the modifications performed by the transaction.
(ii) Replace such values with the values of the data items before the transaction which could either be previously committed or consistent values.

3) ISOLATION [Vimp]

According to this property of transactions isolation must be guaranteed across the execution of transactions in the database.

→ i.e. any transaction will not be able to effect any other transaction during the execution.
→ i.e. All the transactions are guaranteed to be executed in an isolated environment.

* SCHEDULE

- Schedule is the representation of execution of the individual instructions of transactions.
- Schedules are of two types:
  (i) Serial Schedules
  (ii) Concurrent Schedules.

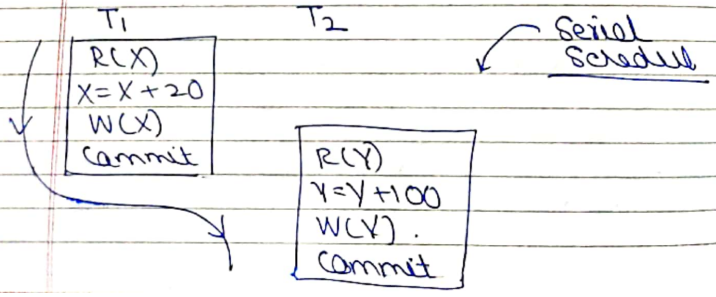
(i) Serial Schedules

- A Schedule is said to be Serial if & only if all the instructions of all the transactions gets executed non-preemptively.

NOTE: If there exists n no. of transactions, then the total no. of possible Serial schedules are n!.

eg: if there exists 3 trans: $T_1 T_2 T_3$.
total no. of possible Serial schedules are: 3! ⇒ 6

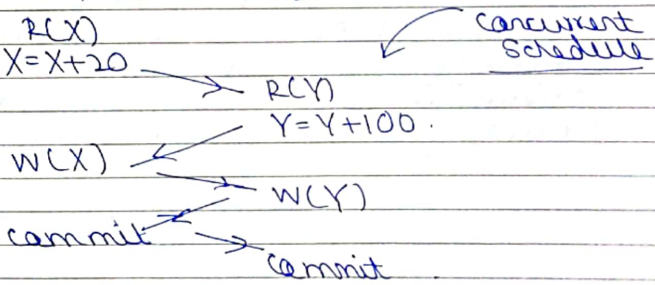
3 → x 2 → x 1 ✓



- Each & every possible Serial Schedule is guaranteed to produce data consistency always.
- Serial Schedules could get affected with:
  (1) Possible high avg. wait time.
  (2) Not acceptable response time.
  (3) Low throughput.

(ii) Concurrent Schedules

- A Schedule is said to be concurrent if the instructions of the transactions are getting executed preemptively. ✓
- Concurrent Schedules guarantees:
  (i) Reduced Avg wait time.
  (ii) Acceptable response time.
  (iii) Optimal throughput.
- However, they can't guarantee data consistency due to the possible conflicting operations.



* Conflicting Operations

A pair of operations are said to be conflicting with each other iff:

(i) they belongs to two different transactions.

(ii) Both the operations are accessing the same data item.

(iii) Atleast one of those operations is a "Write" operation.

	T_i	T_j	
1)	W(X)	W(X).	W → W conflict ⇒ Lost upd. Problem
2)	W(X)	R(X)	W → R conflict ⇒ Dirty read Problem
3)	R(X)	W(X)	R → W conflict ⇒ unrepeated read Prob

* RDBMS implemented the concept called concurrency control for guarenting data consistency always.

– Concurrency control mechanism ensures resulting serializable Schedules always.

* Serializable Schedule:

A Given concurrent Schedule is said to be Serializable if and only if, it produces the result same as atleast 1 of the possible Serial Schedules always.

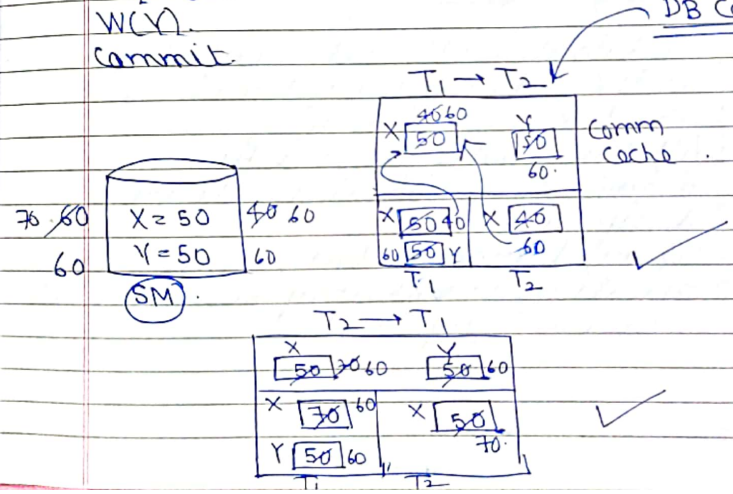
1)

T_1	T_2
R(X)	
X = X − 10	
	R(X)
	X = X + 20
W(X)	
R(Y)	
	W(X)
	Commit.
Y = Y + 10	
W(Y)	
Commit	

	X	Y
$T_1 \rightarrow T_2$	60	60
$T_2 \rightarrow T_1$	60	60
Concurr.	70	60

∴ Not Seriaiz.

W → W conflict [Lost Upd. Prob].



Bless You

concurrent

X 50 ~~50~~ 70	Y 50 60
X 50 40	X 50 70
Y 50 60	
T_1	T_2

NOTE.

1. The Given Schedule is not Serializable.
2. In the above Schedule, the update performed by T_1 on the data item X was getting lost to the update performed by T_2. Hence, the problem is called Lost Update Problem.

Dirty Buffer

- The data item which got modified only in the cache and not yet modified in the disk is refered as dirty buffer.
- All the updates performed by any ongoing transaction ideally, is the dirty buffers.
- If any transaction does perform read operation on such dirty buffer, then its refered as "Dirty Read".

	X
$T_1 \rightarrow T_2$	110
$T_2 \rightarrow T_1$	110
concurrent	100

Disk: 50

T_1	T_2
50 R(X)	
40 X = X − 10	
40 W(X)	
	R(X) 40
	X = X + 60 100
	W(X) 100
	Commit
Rollback	

WR Conflict. "Not Dirty Read Prob".

- The above Schedule is not serializable.
- In the above Schedule, T_2 had been performing the read on the data item that got modified by another ongoing transaction T_1.

Hence, thats referred as dirty read.

T_1	T_2
50 R(X)	
	R(X)
	X = X + 50
	W(X)
100 R(X).	

R_W conflict

unrepeatable read Prob.

[Not Serializable]

Cache [A 10, B 30

T_1	T_2
R(A)	
A = A − 10	
W(A)	
R(B)	
B = B + 10	
W(B)	
	R(B)
	B = B − 20
	W(B)
	R(C)
	C = C + 20
	W(C)
commit	
	commit

A = 20, B = 40, C = 60

20 → 10, 20 → 30, 10

20, 80.

10.

30.

(i) Identify all poss. conflicts

(ii) Is schedule strict / not

20, 40. 60

1) W(B) → R(B) [Dirty] } 3
2) W(B) → W(B) 10 20.
3) R(B) → W(B)

30. 80.

(iii)

	A	B	C
$T_1 \rightarrow T_2$	10	30	80
$T_2 \rightarrow T_1$	10	30	80
Concurrent	10	30	80

→ Serializable Schedule

NOTE:

1. The given schedule has all the possible conflicting operations. Inspite of that, it is producing the result same as the possible Serial Schedules. Hence, the given schedule is declared as Serializable Schedule.

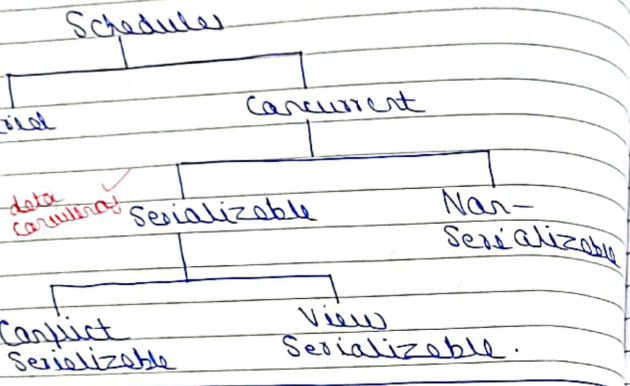
2. The above example proves that if there exists conflicting operations in a schedule, it only means there could be possible data inconsistency.

NOTE: If a given concurrent Schedule becomes equivalent to one of the Serial Schedules, then it is referred as Serializable Schedule.

- If the given concurrent Schedule is conflict equivalent to one of the possible Serial schedules then it is referred as conflict Serializable schedule.
- If the given concurrent Schedule is view equivalent to one of the possible Serial Schedules, then the given schedule is declared as View Serializable.

→ If the schedule is either conflict / view or both, we declare that schedule as ~~conflict~~ Serializable.

→ If it is neither of them, then it is declared as not Serializable.



* Conflict Equivalence

2 Schedules S_1 and S_2 are said to be conflict equivalent to each other if & only if the order of conflicting operation is same in both the schedules.

* Precedence Graph Algorithm

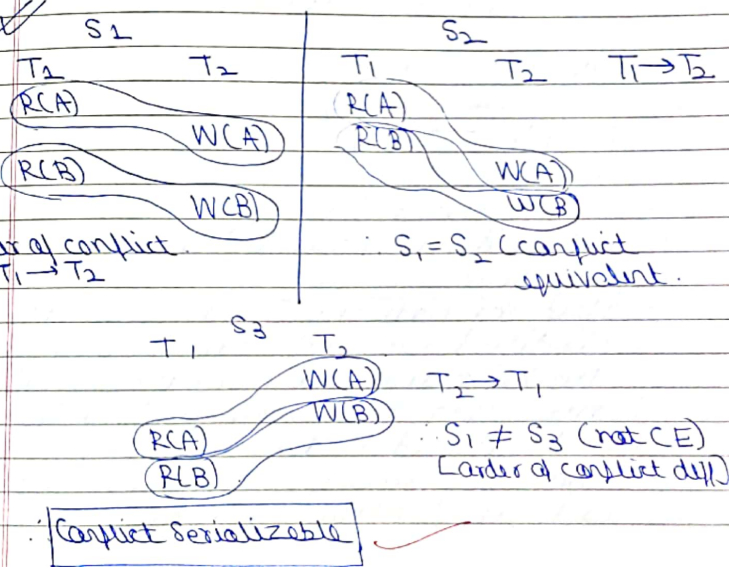
We could apply the algo for finding whether the given schedule is conflict serializable or not.

Step 1: Create the no. of nodes in the Graph that are equivalent to the no. of transaction in the Schedule.

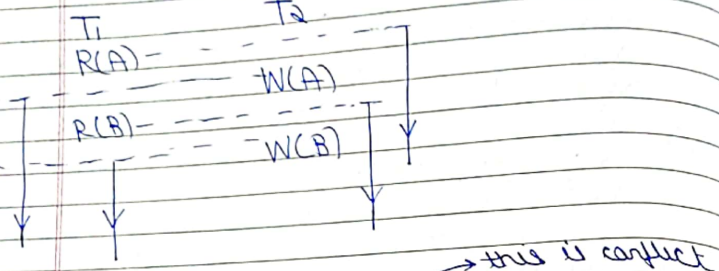
Step 2: Starting from each & every node i.e. each & every transaction, identify all the existing conflicting operations & represent them in the Graph in the form of edges following the direction of conflicting operations.

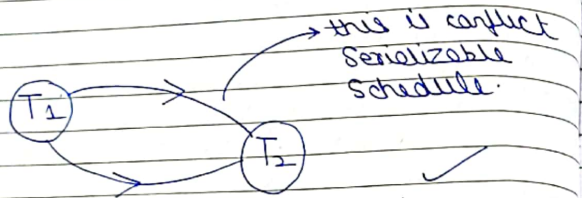
Step 3: Check if the precedence graph has any cycle / loop.
- If the cycle / loop doesn't exist; then the given schedule is conflict Serializable.

Step 4: If the schedule is conflict serializable then apply the topological ordering on the Graph for finding the equivalent Serial Schedule.



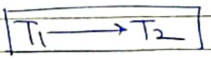
* Prec Graph Algo





Conflict Serializable : no loop

⇒ Topological ordering :



- The process of identifying the node with indegree 0 always, is called topological ordering
- After the node with Indegree 0 gets identified, we must ignore that node further & also we must ignore all the edges that were emerging from that selected node.

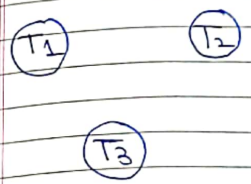
→ From the remaining graph, we must continue identifying the next node

with indegree 0.

→ The above process will continue until all the nodes of the Graph gets selected.

* Take these precedence Graphs :

1)

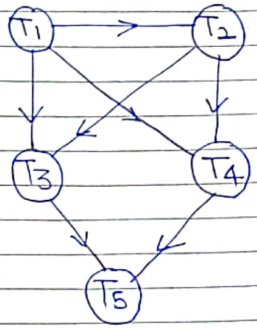


all are ID = 0.

- $T_1 \rightarrow T_3 \rightarrow T_2$
- T_1 ...
- :

Choose any : 3! ways.

2)



$T_1 \rightarrow T_2 \rightarrow T_3 \rightarrow T_4 \rightarrow T_5$

$T_1 \rightarrow T_2 \rightarrow T_4 \rightarrow T_3 \rightarrow T_5$

* If there exists more than 1 node with indegree 0 in the Graph, then we should consider all the possibilities with respect to the equivalent Serial Schedules.

GATE 2015

There exists Schedules as given below, Find whether those are conflict Serializable.

S1

T2	T3
R(X)	
	R(X)
W(X)	
R(X)	
	W(X)

S2

T1	T2	T3
R(X)		
		R(X)
		W(X)
W(X)		
	R(X)	

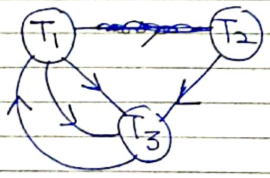
S3

T2	T2	T3
		R(X)
	R(X)	
		W(X)
R(X)		
W(X)		

S4

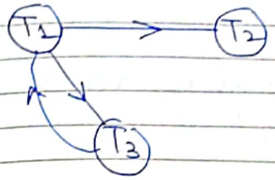
T1	T2	T3
		R(X)
	R(X)	
R(X)		
		W(X)
W(X).		

S1



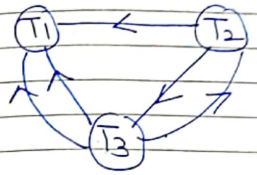
X.

S2



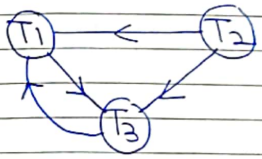
X.

S3



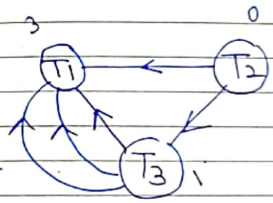
X.

S4



X.

S3



Conflict Serializable

$T_2 T_3 T_1$

[equivalent Serial Schedule]

∴ this is conflict Serializable, hence its also Serializable.

GATE 2007

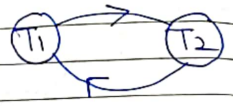
There exists Schedules as Given below. Find out which of them are conflict Serializable.

S1) $R_1(X), R_1(Y), R_2(Y), R_2(X), W_2(Y), W_1(X)$

S2) $R_1(X), R_1(Y), R_2(Y), R_2(X), W_2(Y), W_1(X)$

S1

T_1	T_2
R(X)	
R(Y)	
	R(Y)
	R(X)
	W(Y)
W(X)	

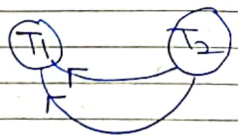


✗ [Not conflict Serializable]

T_2T_1 [Equiv. Serial schedule]

S2

T_1	T_2
R(X)	
	R(X)
	R(Y)
	W(Y)
R(Y)	
W(X)	



✓ conflict Serializable.

* View Equivalence → NP Complete Problem.

2 Schedules S1 and S2 are said to be view equivalent to each other if and only if all the following points are valid for all the data items:

Step1: The transactions that are performing the initial reads on the data item in schedule 1 are expected to perform the initial reads on those respective data items in Schedule 2 as well.

Step2: If there exists [Write → Read] conflict on any data item in S1, then the same conflict must exist on the same data item in S2 as well.

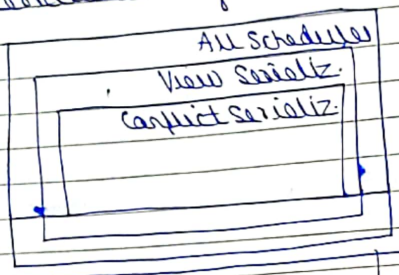
Step3: The transactions that are performing the final write operations on the data items in S1 are expected to perform the final write operations on those respective data items in S2 as well.

NOTE:

1. Finding whether the given Schedule is view Serializable or not is NP complete problem.
2. If a Schedule is conflict Serializable, always it must be View Serializable as well.
   But is not vice-versa.

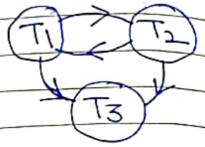
(c) V

* Hierarchy of Schedules based on Serializability:



Q1

T_1	T_2	T_3
R(X)		
	W(X)	
W(X)		



Not CS

Check for VS → No write Read conflict.

View Equiv.

$T_1 \to T_2 \to T_3$	✓
$T_1 \to T_3 \to T_2$	X
~~$T_2 \to T_1 \to T_3$~~	~~✓~~
$T_2 \to T_1 \to T_3$	X
$T_2 \to T_3 \to T_1$	X
$T_3 \to T_1 \to T_2$	X
$T_3 \to T_2 \to T_1$	X

all Serial equivalents

VS ✓

initial Read = T_2, T_3] X var.
final write = T_1.

Q1

T_1	T_2	T_3
	R(X)	
		R(X)
		W(X)
R(X)		
W(X)		

→ conflict Seri ✓

init val = $T_2 T_3$.
last = T_1

* Check for VS

$T_1 \to T_2 \to T_3$	X
$T_1 \to T_3 \to T_2$	X
$T_2 \to T_1 \to T_3$	X
$T_2 \to T_3 \to T_1$	✓
$T_3 \to T_1 \to T_2$	X.
$T_3 \to T_2 \to T_1$	X.

T_2	T_3	T_1
R(X)		
		R(X)
		W(X)

View Serializable and conflict Serializable. ✓

WR order $[T_3 \to T_1]$ same.

Q2

T_1	T_2	T_3
R(X)		
		R(X)
W(X)		
		W(X)
R(X)		

$T_2 \to T_1 \to T_3$
$T_2 \to T_3 \to T_1$

T_2	T_1	T_3
R(X)		
	R(X)	
	W(X)	
		R(X)
		W(X).

Not VS

→ initial read
final write = T_3.

T_1	T_2	T_3
R(X)		
		R(X)
W(X)		
		W(X)
	R(X)	

→ 2 initial reads [T_1, T_3]

if we start with T_1, T_3 want satisfy initial read.

∴ No Serializability

Note: If there exists more than 1 transaction performing the initial read on the data item and atleast 2 of those transactions are performing the write on the same data item, then the scheduled will never be view Serializable.

Q

T_1	T_2	T_3
		R(X)
	R(X)	
R(X)		
		W(X)
W(X)		

{T_1, T_2, T_3} will get initial read.

Not VS

⇒ Not Serializable.

4) DURABILITY

According to this Property of transact, all the modifications performed by any committed transaction are ensured to be present in the Non-Volatile Storage of data Base.

⇒

* Write Ahead log Protocol

According to this protocol, all the modifications performed by each & every committed transaction, must & should be written to a special file called redo log file. before they gets written to the actual data file.

* Redo log file → used by only RDBMS.

- It is the special file used by only RDBMS during the recovery of the data base.
- This file consists of collection of Redo log buffers.
- Each & every redolog buffer consists of the following information:
  (1) Copy of the data items before the transactions.
  (2) Copy of the data items after the transactions.

[Shadow Paging by Redo log buffer].

(3) Details of the transaction that perform the modification.

* RECOVERY

According to ARIES database Recovery Procedure, RDBMS gets into database recovery in 3 phases.

These are:

(1) Analysis Phase
(2) Redo / roll Forward Phase.
(3) Undo / roll Back Phase.

1) ANALYSIS PHASE

- During this Phase of recovery, RDBMS will calculate the total no of transactions that shall get recovered.
- In each & every RDBMS, there exists an unique no associated with each & every committed transaction.

eg: In oracle, that is System Change no.
SCN #
In MSS, that is Log Sequence no.
LSN #

2) Redo / ROLL FORWARD PHASE

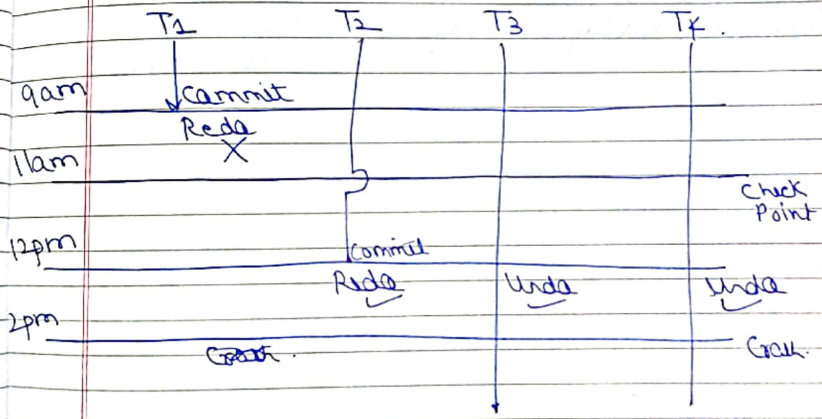
During this Phase of recovery, RDBMS will start reapplying / reading all the entries of redo log file irrespective of the entry belongs to a committed transaction or Not.

- all transactions in day, must be present in redo log file.

3) Undo / Roll Back Phase

- During this Phase of recovery, RDBMS will perform abort operation on the entries belongs to uncommitted / open transactions.

Diff b/w com



[Shadow Paging Utilized by Redo Log buffer].

(3) Details of the transaction that perform the modification.

* RECOVERY

According to ARIES database Recovery Procedure, RDBMS gets into database recovery in 3 phases.

There are:
(1) Analysis Phase
(2) Redo / roll Forward Phase.
(3) Undo / roll Back Phase.

1) ANALYSIS PHASE

- During this Phase of recovery, RDBMS will calculate the total no of transactions that shall get recovered.
- In each & every RDBMS, there exists an unique no associated with each & every committed transaction.

eg: In oracle, that is System Change no. SCN#
In MSS, that is Log Sequence no. LSN#

2) Redo / ROLL FORWARD PHASE

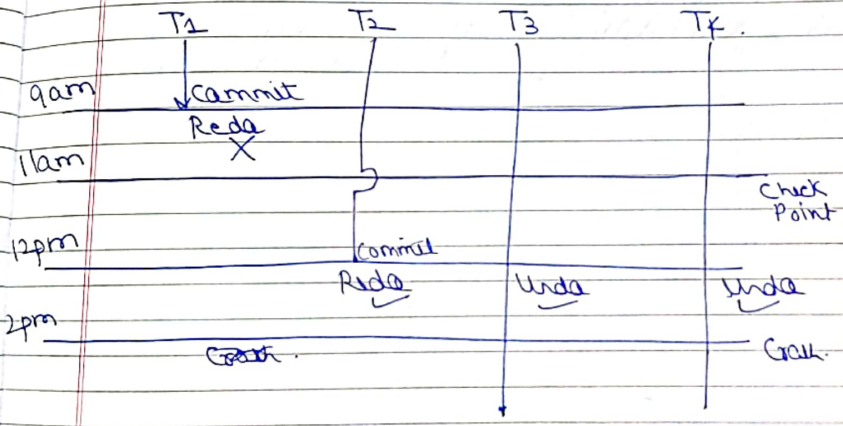
During this Phase of recovery, RDBMS will start reapplying / reading all the entries of redo log file irrespective of the entry belongs to a committed transaction or Not.

- all transactions in day, must be present in redo log file.

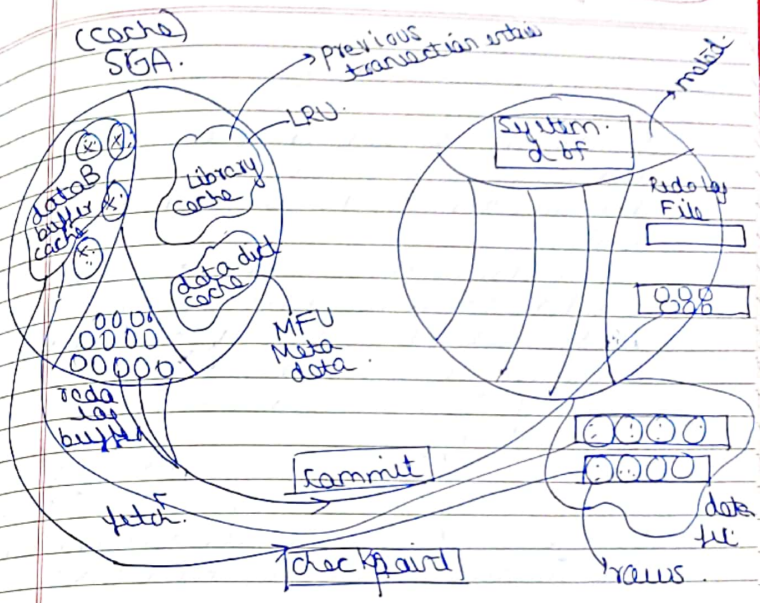
3) Undo / Roll Back Phase

- During this Phase of recovery, RDBMS will perform abort operation on the entries belongs to uncommitted / open transactions.

Diff b/w com



Commit	Check Paint.
(ii) → This is a transact Specific command	This is a server wide operation
(ii) When commit is issued, all the intern belong to that transact are guaranteed to be transferred to the redo log File.	(ii) When it is issued all the dirty buffers across the DB Server are assured to be transferred to the data File. If the transaction gets committed before the check Point, we need not apply Redo operation on such transaction.
(iii) ~~If a transaction gets~~	
(iii) This command is issued for guaranteeing the durability Property.	This command is used for reducing the time of database recovery.



→ update emp
set sal = sal + 500.
where deptno = 20
→ Commit.

* SCHEDULES BASED ON Recoverability

- Based on recoverability, Schedules are divided into 3 types:
(i) Recoverable Schedule.
(ii) cascadeless Schedule.
(iii) Strict Schedule.

(ii) Recoverable Schedule

A Schedule is said to be recoverable if and only if, for each & every pair of transactions (T_i, T_j) in such a way that:

T_i performs the modification on the data item followed by T_j reads that; then always completion of the transactions should be in the order of T_i followed by T_j.

(T_i → Parent [Master], T_j → child [dependent])

→ Recoverable Schedules gets affected by the problem of cascading RollBacks.

* Cascading RollBacks

In case if Parent/Master transaction gets rollBack, then to maintain data consistency all the subsequent child/dependent transactions also should get rollbacked.

→ This problem would cause lots of redo Activity.

T_i	T_j
R(X)	
X=X+50	
W(X)	→ Dirty read.
	R(X)
	X=X+100
	W(X)

(iii) Cascadeless Schedule

(always recoverable (not vice-versa))

- A Schedule is said to be cascadeless iff each & every transaction is allowed to perform read operation on the data items updated by only the completed transactions.
- dirty read never allowed.

NOTE:

(i) All the cascadeless Schedules are recoverable but not vice-versa.

(ii) Cascaded Schedules will restrict the possible dirty Read Problem but, it cannot restrict the Potential write-write conflict (Lost Update Prob)

(iii) Cascadeless Schedules could have possible data Inconsistency.

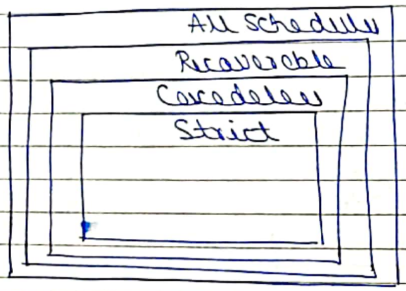
(iii) Strict Schedules

A Schedule is said to be strict iff ~~every~~ any transaction is neither allowed to perform read nor to perform update on the ~~to~~ modification done by other transaction, until other transaction gets completed.

NOTE:

1. All the Strict Schedules are always cascadeless & recoverable but not vice-versa.
2. SS would control the concurrency of execution.
3.

* Hierarchy of Schedules based on recoverability

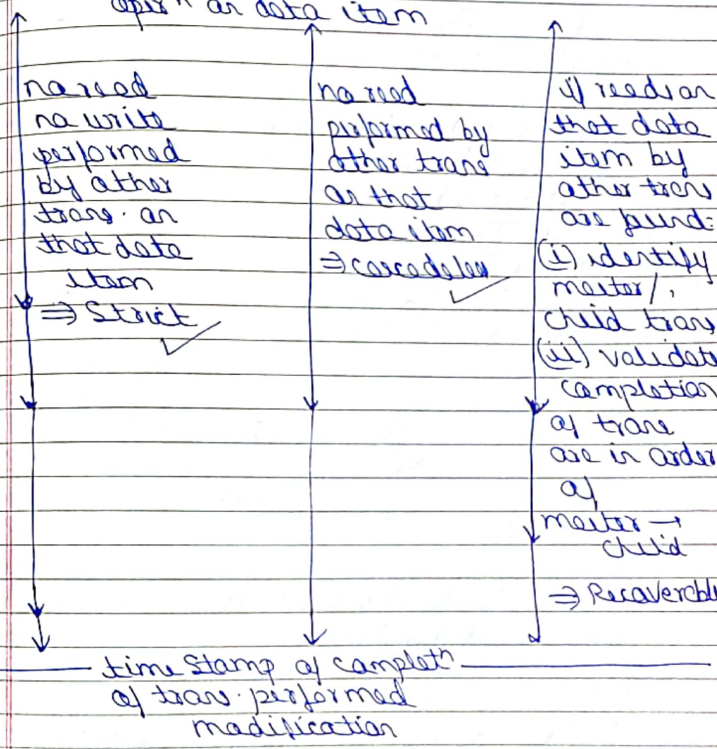


** Procedure for finding Highest Recoverable Option for Given Schedule.

Step 1: Find out all data items present in the Schedule.

Step 2: For each data item, perform following Steps:

S-3: Identify the time Stamp of modification performed on data item and completion of transaction that performed modification.



S-4: Highest common recoverable for data items
⇒ Highest recovery for Schedule.

Q) Then exits Schedule as represented:

$R_2(A), R_3(A), R_1(A), W_1(B), R_2(B), R_3(A), W_2(C), R_3(C), A_1$

Abort for T_1.

T_1 T_2 T_3

R(A) R(A)

- If T_1 gets aborted, which of the other transactions also will get rollBacked?

a) only T_2
b) only T_3
c) both T_2, T_3
d) Neither T_2, T_3.

T_1 T_2 T_3.

R(A) R(A)

R(A)

both T_2, T_3 should rollback.
∵ dependent.
[cascading RB]

Q) Find the highest recoverable option for the following Schedules.

S1) $R_1(X), R_2(Z), R_1(Z), R_3(X), R_3(Y), W_1(X), C_1, W_3(Y), C_3, R_2(Y), W_2(Z), W_2(Y), C_2$

S2) $R_1(X), R_2(Z), R_1(Z), R_3(X), R_3(Y), W_1(X), W_3(Y), R_2(Y), W_2(Z), W_2(Y), C_1, C_2, C_3$

S3) $R_1(X), R_2(Z), R_3(X), R_1(Z), R_2(Y), R_3(Y), W_1(X), C_1, W_2(Z), W_3(Y), W_2(Y), C_3, C_2$

CL.

S1) $X \rightarrow$ Strict. $Y \rightarrow S$. $Z \rightarrow S$. } Strict

S3) $X \rightarrow S$ $Y \rightarrow C$ $Z \rightarrow S$ } cascadeless. → Not Recov

S2) $X \rightarrow S$ $Y \rightarrow$ RNR. $Z \rightarrow$ → $T_3 \rightarrow T_2$

T_3 complete first.
But in Q: T_2.

13/10/19
Sunday

3rd

classmate Date Page 102

Lecture 4

* CONCURRENCY CONTROL

- RDBMS has concurrency control implemented through locking mechanism for ensuring data consistency always.
- RDBMS has a sep. prog called lock manager which is responsible for either accepting/rejecting the lock request of different transactions.
- Prog listens to lock requests.

→ Lock Manager internally makes use of the lock compatibility matrix while taking all the decisions
(existing state of item & req state)

→ Initially, there exist only 2 modes for the data items. Those are:
(i) locked mode
(ii) unlocked mode.

As noticed, the degree of utilization getting reduced, they introduced 2 different modes of locked states. Those are:
(i) Shared mode
(ii) Exclusive mode.

→ If in case the transaction wants to perform only the read operation, then they could acquire Shared mode of lock on the data item.

→ In case if the transaction wants to perform both read & write opn, then they should acquire an exclusive mode of lock.

→ The concurrency control mechanism developed in RDBMS would aim for producing the strict serializable schedules always.
(from recov. / from consist.)

→ Transaction prod. lock req.

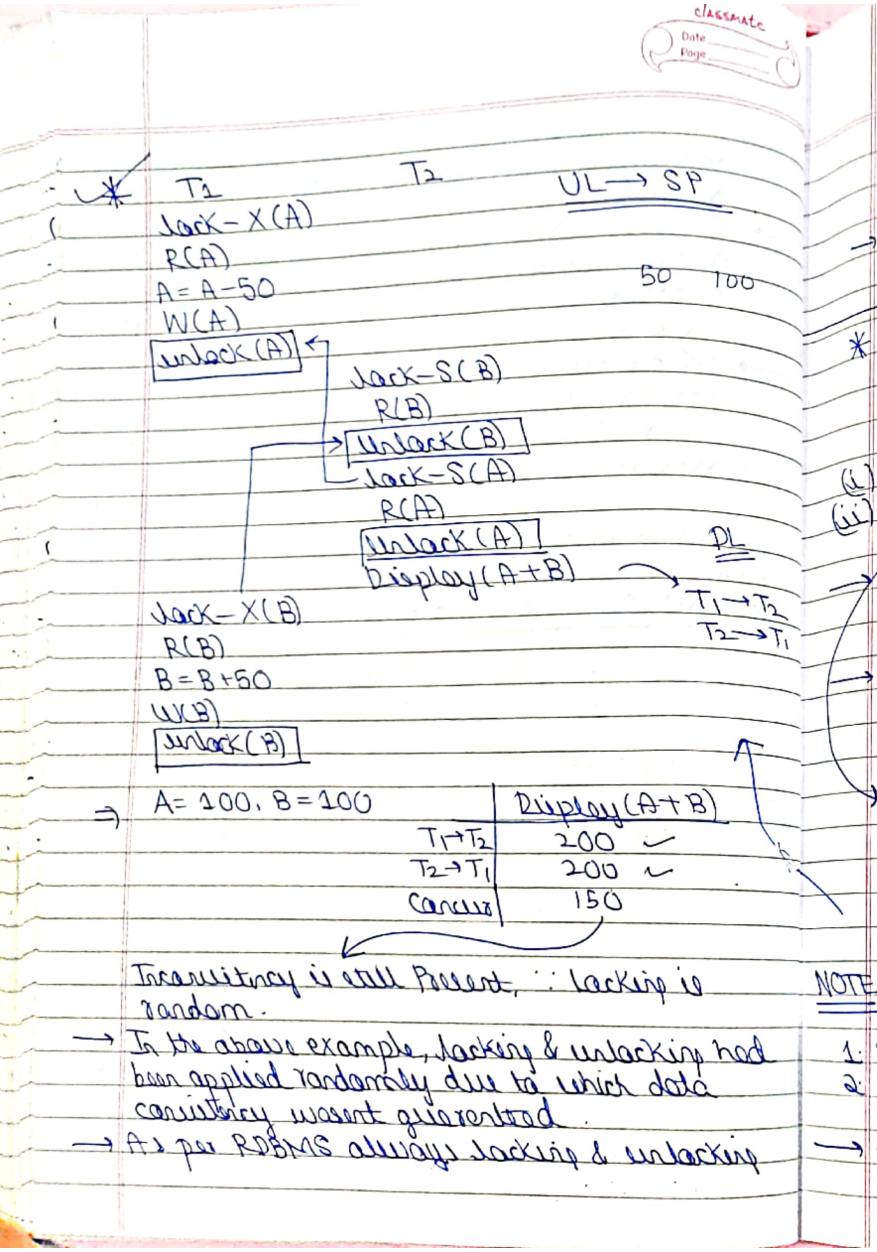
classmate Date Page 103

* Lock compatibility Matrix

	Current Mode		
Req Mode	Unlock	S-Mode	X-Mode
Unlock	—	Yes	Yes
S-Mode	Yes	Yes	No
X-Mode	Yes	No	No

S not comp. with X.

T_1 wants to read, T_2 is writing ∴ T_1 needs to wait in Que



Inconsistency is still Present, ∵ locking is random.

→ In the above example, locking & unlocking had been applied randomly due to which data consistency wasn't guaranteed.

→ As per RDBMS always locking & unlocking should be performed as per a protocol called 'Locking Protocol'.

→ One such widely used 'Locking Protocol' is 2 Phase Locking Protocol.

* 2 PHASE LOCKING PROTOCOL

According to this Protocol, the transaction process will be done in 2 Phase. These are:

(i) Growing Phase
(ii) Shrinking phase.

→ During the Growing Phase all the transact. are allowed to acquire all the locks required for its processing.

→ Transactions cannot afford to release any of the locks acquired in this Phase.

During this Phase of transaction Process, the transactions are allowed to release all the acquired locks & they cant afford to acquire any new lock for them.

NOTE: 2 Phase LP gets affected by:

1. The deadlock Problem.
2. Cascading rollback Problem. → if P rollbacks, all child rollbacks.

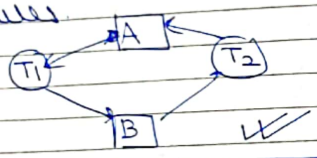
→ Deadlocks would be addressed by detection & recovery techniques.

⇒ RDBMS makes use of Resource Allocation Graph (RAG) for detecting the DL.

· As part of Recovery:

· one of the transaction can be selected as victim & kill the transaction.

· 2 Phase LP always ensures the serializable Schedules.



* Strict 2 Phase Locking Protocol

- This is higher variant of 2P LP in which Growing & Shrinking Phase concepts are still valid.
- In addition to them, a new rule of "not releasing all the acquired ~~locks~~ exclusive locks of the transaction gets added".
  · can release till T_i commits.
- This Protocol also gets affected by the Deadlock Problem.
- This protocol always Produces the strict Serializable Schedules.

TIMESTAMP BASED PROTOCOL

→ The most unique concurrency control technique that doesn't have the concept of lock & Deadlock is time Stamp based Protocol.

→ As per this protocol, an unique TS will be assigned for each & every transaction.

For each data item, 2 TS gets assigned.

There are:

(i) read TS

(ii) write TS.

→ The TS of the transaction that performed the most recent read on a data item becomes the read TS of the data item.

→ The TS of the transaction that performed the most recent write operation on a data item becomes the write TS of the data item.

→ TS based Protocol is not implemented in the SQL DataBase practically yet.

* ER DIAGRAMS

* Phase of database Design:

1. Requirement Analysis:

During this phase of DB design, users will gather all the requirements for creating the database.

2. Conceptual Design

During this phase of DB design, ER diag would get prepared. Preparing the ER diagram is independent of the RDBMS selection, OS platform selection, Geography etc.

3. Logical Design

During this phase of dB design, we would decide all the different tables would be part of the DB & the list of columns that would be part of each table along with list of integrity constraints.

4. Physical design

During this phase of DB design, users would execute: create DB, create table commands.

* Referential Integrity constraints:

- This is the process of validating the correctness of data with a reference.
- Foreign Key constraints would eventually validate the referential integrity of data.

* Foreign Key

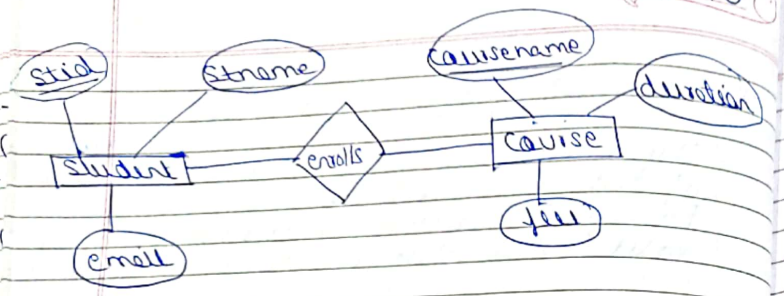
- The Primary Key of Master/Parent Table which would exist in the child/dependent tables is referred as Foreign Key in the child Table.
- while defining the foreign key, there exists 2 options:
  (i) On delete cascades.
  (ii) On delete No Actions.

— must be PK of other relation.

→ FK always guarantees the parent-child relationship.

Imp) According to referential integrity/parent-child relationship, a value is allowed in the foreign key column of the child Table only if it has a reference to one of the values of the Primary Key column of the Parent Table.

- It is not vice-versa.



PK Student FK

Stid	Stname	email	Cname
100	XYZ	–	ML

parent

checks all the rows in PK of Parent

PK

Cname	Dur	Fee
SAP		
Azure		
AWS		
ML		

→ only one PK cannot accept CK.
∵ the rows of that col should get sorted
decision by business idea.

1 PK

* Relations are allowed to have many CK. out of them only 1 PK is allowed because the decision of PK creates a clustered index internally. That is all the rows of the table will get sorted automatically based on the values of PK column by default.
→ helps in Binary Search.

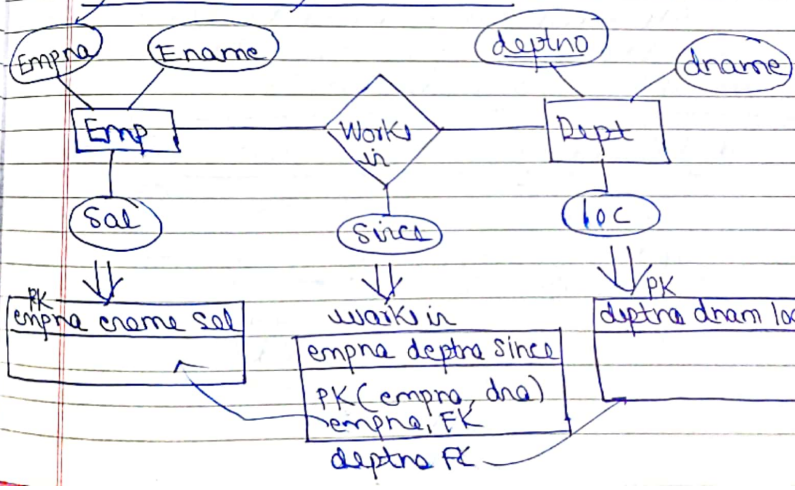
(ii) On Delete Cascade

If the FK is defined with this option, then in case if the PK value gets deleted then By default all the referencing FK values of the child table also are allowed to be deleted for validating the referential integrity.

(iii) On Delete No Action

If the FK is defined with this option, then user have to manually perform update operation on the FK values & ensure there are no references to the PK value that is going to get deleted.

* BASIC ER DIAGRAMS



* The entities can part. in relation with three cardinalities :

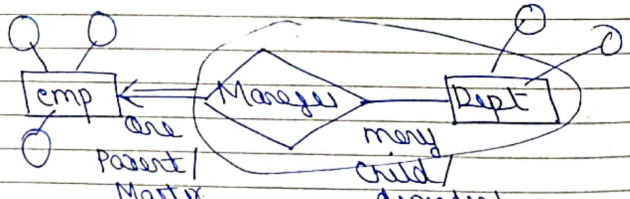
1. One-One relationship.
2. One-Many relationship
3. Many-One relationship
4. Many-many relationship.

- In case, cardinalities are represented in ER diagram, it becomes easy to identify the parent & child entities.
- We could minimise the no. of tables for representing the ER diag by combining the child/dependent entity along with the relationship.

* Descriptive Attributes

Attributes that belong to the relationship only and they doesn't belong to any of the entities participating in the relationship.

* ER Diag. with Cardinalities



emp
empno (PK)
Ename
Sal

Dept
deptno (PK).
dname
loc
since
empno (FK), NOT NULL

Parent → single.

NOTE: Entities could participate in the rel. in 2 possible ways. Those are:
(i) Total/complete Participation.
(ii) Partial/incomplete Participation.

In case, if the participation is total/complete then FK values should be not NULL in the child tables.
- otherwise FK values could be NULL.

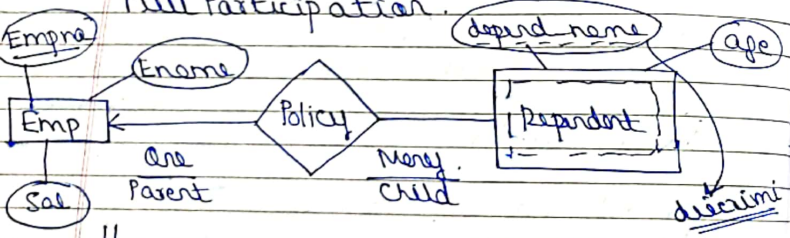
⇒ Total/complete participation would be represented by either a double/double line.
⇒ Incomplete/Partial particip. is represented by a dashed line.

- By default, all the entities are expected to participate in the relation fully as totally.

* Weak Entities

- The entities which doesn't have any PK is called weak entity.
- weak entities always involves in a relationship with other strong entities for identifying the rows inside weak entity.
- The column/set of columns of weak entity that gets combined with primary key of the strong entity for the identification of rows is called discriminator.

→ Discriminators are represented with underlying the attributes with a dashed line.
→ Weak entities could only involve in the relationship with other Strong entities in Full Participation.



Emp
Empno (PK)
Ename,
Sal

Dependent
depend_name
age
empno FK NOT NULL
PK(empno, dependentname)

GATE Qs

Gate 2006.

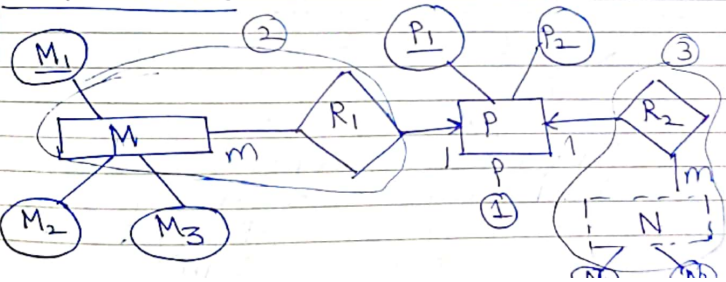
There exists relation R, with A & C attributes.
A ⟶ PK.
C ⟶ FK that references A with an delete cascade property.
If tuple (2,4) gets deleted, which other tuples should also get deleted as per referential integrity.

	PK A	C FK
⇒	2	4
	3	4
	4	3
⇒	5	2
	6	4
⇒	7	2
⇒	9	5

on delete cascade.

(d) {(5,2), (7,2), (9,5)} also gets deleted.

GATE-2008



⇒ 3 Tables minimum.

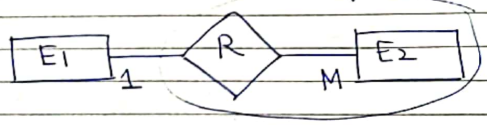
P
P_1 PK
P_2

M
M_1 PK.
M_2
M_3
P_1 FK

N
N_1
N_2
P_1 FK NOT NULL.
PK(P_1, N_1); ∵ N_1 → discriminator.

Gate 2004

There exists ER diag given below along with cardinalities represented.



∴ 2 min Tables

FK
R_1(A, B) ; R_2(C, D)
PK PK

Gate 2012

There exists relations R_1 & R_2 with (A, B) & (C, D) attributes respectively. Let us

R_1 (A, B) R_2 (C, D)

assume A and C are the PK in those relations and B is the FK that references to C in R_2.
In case if all the data satisfies, the referential integrity concept then which of the following statements always.

a) $\Pi_C(R_2) = \Pi_B(R_1)$

b) $\Pi_C(R_2) - \Pi_B(R_1) = \phi$

c) $\Pi_B(R_1) - \Pi_C(R_2) = \phi$ → in R_1, but not in R_2

d) None.

B can't have any values which isn't there in C.

Gate 2012

There exists a table T with (X, Y) attrib. A row got inserted into the table with the values (X=1, Y=1).
Let MX, My denote the highest values of X & Y at any given point in time.
Now rows are inserted into the table with the values:

X = MX + 1
Y = 2My + 1

Then find out the result of the Query?

X	Y
1	1
2	3
3	7
4	15
5	31
6	63
7	127

$Y = 2^X - 1$

Q There exists train reservation system as follows: [Gate 2010]

Passenger

Pid	Pname	Age
0	Sachin	65
1	Rahul	66
2	Sourav	67
3	Anil	69

Reservation

Pid	class	Tid
0	AC	8200
1	AC	8201
2	SC	8203
3	AC	8204
5	AC	8206
1	SC	8202

```
Select R.Pid
from Reservation
where class='AC'
    and
    exists( Select *
            from Passenger P
            where Age>65
                and
                P.pid=R.pid).
```

Result: (1/3).

(d) (1,3)

Q Gate 2005

There exists a relation Book with (Title, Price) info.
Let us assume the relation has info of atleast 6 books.
The title & price in the table are supposed to be unique.
What will be the o/p of query?

```
Select B.Title
from Book as B
where ( Select count(*)
        from Book as T
        where T.price > B.price) < 5
```

Soln

B & T

Book

Title	Price
A	500
B	450
C	400
D	150
E	100
F	200
G	250
H	300

CR end to NR

o/p: A, B, C, G, H

⇒ 5 Most expensive Books

Gate-2011

Loan-Records table Given below:

Borrower	Bank-Mgr	Loan-Amount
Ramesh	Sundar	10,000
Suresh	Ramgopal	5000
Mahesh	Sundar	7000.

```
Select Count (*)
from ( (Select Borrower, Bank_Mgr
        from loan Records) As S
        NATURAL JOIN
       (Select Bank_Mgr, loan_Amt
        from loan_Records) as T.
)
```

S

R	S
S	Ramgopal
Mahes	Sudar

T

Boo B	L.
Sundar	1

S

Ram	Sundar
Suresh	Ramgop.
Maheu	Sundar

T

Sundar	10K
Ram	5K
Sundar	7K.

⇒ 5 Rows.

(S) Sailors (Sid, Sname, age, Grade).

(B) Boats (Bid, Bname, color).

Reserves (Sid, Bid, day).

(R)

1) $\Pi_{sname}\left[\sigma_{\substack{S.Sid=R.Sid \\ \wedge \\ R.Bid=B.Bid \\ \wedge \\ B.color='Red'}} (S \bowtie B \bowtie R)\right]$ } NT.

distinct Sailor Names who reserved at least one red color boat

$\cup$

$\Pi_{sname}\left[\sigma_{\substack{ \\ \wedge \\ B.color='Green'}} (S \bowtie B \bowtie R)\right]$

ii):

(ii) Union : distinct Sailor names who reserved at least one red boat or atleast one green

(ii) Intersection: at least one red & at least one Green.

(iii) difference : atleast one red colour but didn't reserve atleast one green.

$$\rho_{temp}\left[\Pi_{sid,bid}(R) - \Pi_{Bid}\left(\sigma_{color='Red'}(B)\right)\right]$$

Bid ⊆ (Sid, Bid)

$$\Pi_{sname}\left[\sigma_{s.sid=temp.sid}(S \bowtie temp)\right]$$

→ distinct Sailors names who have reserved all red colour Boats.

GATE 2003

There exists relations as given below in the database

S - Students (rollno, name, d_o_b)
C - Courses (courseno, coursename, Instructor)
G - Grades (rollno, courseno, Grade)

From S, C, G


```
select distinct S.name
from S, C, G
where S.rollno = G.rollno
      and
      G.courseno = C.courseno
      and
      C.Instructor = 'Karth'
      and
      G.Grade = 'A'.
```

→ Distinct Student name who got grade A in atleast one of the courses instructed by Karth

RA

$$\Pi_{S.name}\left[\sigma_{\substack{S.rollno=G.rollno \\ \wedge \\ G.courseno=C.courseno \\ \wedge \\ C.instructor='Karth' \\ \wedge \\ G.grade='A'}}(S \bowtie C \bowtie G)\right]$$

TC

$$\{t \mid \exists S \in Students \wedge \exists C \in Courses \wedge \exists G \in Grades.$$
$$(S.rollno = G.rollno \wedge G.courseno = C.courseno \wedge C.instructor = 'Karth' \wedge G.grade = 'A' \wedge t.name = S.name)\}$$

eliminate duplicate values

Gate 2013

S Students (rollno, Sname)
C courses (courseno, Cname)
R Registration (rollno, courseno, percent)

```
Select S.name distinct
from S, R
where S.rollno = R.rollno
      and
      R.courseno = 107
      and
      R.percent >= 90
```

⇒ $\Pi_{S.Sname}\left[\sigma_{S.rollno=R.rollno \wedge R.courseno=107 \wedge R.percent>90}(S \bowtie R)\right]$ (Same Value as query)

RA

TC $\{ t \mid \exists S \in Students \wedge \exists R \in registration (S.rollno = R.rollno \wedge R.courseno = 107 \wedge R.percent > 90) \wedge t.sname = S.sname \}$

Gate 2001

Q $R(A,B,C,D,E)$
$\{ A \to B,\ BC \to E,\ ED \to A \}$

$A^+ = \{A, B\}$
$BC^+ = \{B, C, E\}$
$ED^+ = \{E, D, A, B\}$
$\Rightarrow \{EDC, BCD, ACD\}$

2NF

$R(L,M,N,O,P)$
$\{ M \to O,\ NO \to P,\ P \to L,\ L \to MN \}$

$M^+ = \{M, O\}$
$NO^+ = \{N, O, P, L, M\}$
$(P)^+ = \{P, L, M, N, O\}$
$(L)^+ = \{L, M, N, O, P\}$
$\{NO\} \to CK$

2NF CK $\{L, P\}$

NO
↑
NM ← L ← P

∴ 3NF

Gate 2003

$R(A,B)$ $\{ A \to B \}$ FD — BCNF
$R(A,B)$ $\{ B \to A \}$ — BCNF

$R(A,B)$ $\{ A \to B,\ B \to A \}$ — BCNF
$R(A,B)$ $\{\}$ — BCNF

$\{A, B\}$.
Assume $AB \to AB$.

NOTE: Binary Relations would always exist in BCNF.

Gate 2013

P P NP NP P P NP NP

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

{

CH → G

A → BC (B→C, B→F, B→H)

B → CFH

E → A

F → EG (F→E, F→G)

}

AD → ED → FD → BD

(CH)+ = {C, H, G}

(A)+ = {A, B, C, F, H, E, G}

(B)+ = {B, C, F, H, E, G, A}

{A, B, E, F,

{AD, BD, ED, FD} → CKS.

∴ 1NF.

R(N, R, C, T, Z)

{

N → RCT FD.

RCT → Z TD.

Z → CT TD

}

RCT+ = {R, C, T, Z}.

N+ = {N, R, C, T, Z}

Z+ = {Z, C, T}.

N → CK.

2NF

R(V, W, X, Y, Z).

{

VY → W

WX → Z

ZY → V

} OCD

VY+ = {V, Y, W}

WX+ = {W, X, Z}

ZY+ = {V, Z, Y, W}

{WXY → ~~W, X, Z, V~~, W, X, Y, Z, V

{~~XXXX~~, {XZY, WXY, VYX}

VYX = {V, Y, X, W, Z}

3NF

R(A, B, C, D, E, F, G)

{

A → B

BC → DE (BC→D, BC→E)

AEF → G

}

A+ = {A, B}

BC+ = {B, C, D, E}.

AEF+ = {A, E, F, G, B}

AEFC = {A, E, F, G, B, C, D, G}

1) Candidate Key for relation R is: ACF

(ACF)+ = {A, C, F, B, D, E, G}.

⇒ ACF.

AEFC.

ABCF.

2) a) 1NF ✓

R(A, B, C, D)
{
AB → CD
B → A
C → B
C → D
}

2NF
BCNF

TD. {AB, B}

SK
(AB)

CK
{B, C}

R(A, B, C, D, E, F)
{
AB → CDEF FD.
C → B
D → C
E → D
F → E
} OCD.

C+ = {C, B}.
D+ = {D, C, B}

⇒ {AB, AC, AD, AE, AF}.
⇒ 3NF

19. (14)

Q R(A, B, C, D)
{
AB → CD
C → A
D → B
}

AB, CB, CD, AD } CK.

⇒ 3NF = Highest NF

* B+ TREES

We use B+ tree for organising the data in the database because:

1. B+ trees are balanced.
2. They exhibit Binary Search tree property
3. In B+ Trees, data of the indexed columns would be organised at the non-leaf level and the actual rows of the table would be organised at the leaf level.
4. The above feature makes B+tree as the best fit for the range based query execution and also the optimal resource utilisation.

* ORDER / DEGREE OF B+ TREE

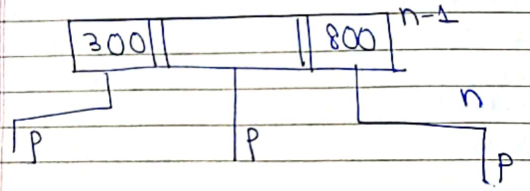
NOTE: The total no. of index/tree pointers that could be available for a B+ tree node.

→ If we assume 'n' is the order/degree of B+ tree then such B+ tree node is expected to have (n-1) no. of index key values.

→ Let us assume:
Each index/tree pointer consume p no. of bytes and the index key value consumes K bytes then the following expression should be valid:

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

B → Block Size of the File.



SQL GATE QS

Q Gate 98 — CSE ✓

- FREQUENTS A
- SERVES B
- LIKES C.
- write the query → Parlour (NJ).

Q Gate 99

a) SQL query cant autom. eliminate duplicate.

b) - index is maintained at Physical Level.
- index used in Query optimizn, not SQL.
- SQL → logical level.

c) 2 attributes with same name not allowed in SQL. ✓

Q3 Select district w, x from r, s.

→ Cartesian Product.

r

w	x
a	b
c	d

s

y	z
1	2
3	4

w	x	y	z
a	b	1	2
a	b	3	4
c	d	1	2
c	d	3	4

a b
c d ✓

but if, w x → result remains different
a b
c d
a b

if S is empty, CP no o/p.
↳ same R not possible.

⇒ r has no duplicates & S is non-empty.

Q Gate 2003

equivalent RA expression.
⇒ CP of all the tables.

a) $\Pi_{a_1, a_2 \cdots a_n} \sigma_p (r_1 \times r_2 \times \cdots r_m)$

b) Join condition is used.

Q GATE 2011 → 5 tuples (Repeated Q)

Q Gate 2012

Nested Query → Not correlated Query.

execute internal Query once, place it & execute the outside Query.

→ Check row by row in outside & table.

- Nothing will be retrieved ∴ 3

Q Gate 2014

employees
departments

only for some departments.

IN → pairwise comparison allowed

→ execute inner query first (non-correlated).

emp_id fname ... sal deptid dname ... id.

1 2 3 4 ... 1700 1710

Emp

deptid	hire_date max
1	1-1-2019.
2	2-1-2019

Q Gate 2014

Select * from R where a IN (Select S.a from S)

R → table, a → attribute.

duplicates allowed.

R	S	S1
a	a	a
1	2	2
1	2	5
2	5	9
2	5	4
3	9	
4	4	

(2,5,9,4)

d/p: 2, 2, 4

2
2
2
2
4
RS
22
22
22 ⇒ [22224]
22
44

a) {22224} X
b) {2,4} X
c) {2,2,4} ✓
d)

Select R.* from R,S, where R.a = S.a and
is implicit R. X.
∴ implicit is used while creating table/
in Select query

Gate 2005

Nested correlated Query

→ Take eg always to run SQL query

↳ 6 books

Book (B)	
T	P
a	1
b	2
c	3
d	4
e	5
f	6

Candidate row →

T	
T	P
a	1
b	2
c	3
d	4
c	5
f	6

all rows are compared with the candidate row

(d) five most expensive books.

Gate 2003

CP in a meaningful way.

(c) atleast one of the courses taught by Karth & got 'A' grade.

Gate 2004

group by deptname.
display only dnames, whose average salary

Not correlated Query

avg of all salaries in company
↳ 100

having → filter out groups

(d) The avg salary of male employees is more than avg salary in company.

Gate 2006

(Student, course) → PK in Enrolled
Student → PK in Paid

Q4) For every row in 'Paid', we need to execute the inner query once. → Correlated Sub Query.

Consider Duplicates

eg →

Enrolled

Student	Course
a	1
a	2
b	1
b	2
c	1
c	2

Paid

Student	Amount
→ a	10
b	11
c	12

Q1) {a, a, b, b, c, c} ∵ non-correlated query.

Q2) {a, b, c} ∵ inner query = a a b b c c.

Q3) {a, a, b, b, c, c}

Q4) Correlated inner query → s/s: a, b, c

(a, 10) →

for (b, 11), outer query is executed → inner Q.

b 1
b 2

exists : these exist atleast one row / not.

(c, 12) →

→ All compute intersection b/w 2 sets.

a) identical row sets. ✗ (no duplicates)

b) ✗

c) ✗ [Q3 → more rows than Q2].

d) ✗ no FK, no integrity violation

Gate 2007

Salary greater than all employees in dep 5

e.

eg →

	emp id	name	dep	sal	
e →	1	x	5	100	← s.
→	2	x	2	200	
→	3	x	5	300	
→	4 ✓		2	400	
→	5 ✓		2	500	

NOT exists → when inner query returns nothing
e. empid will be printed.

1) o/p = {4, 5}.

2)

e		
empid	Sal	dept
→ 1	100	5 X
→ 2	200	2
3	300	5
4	400	2
5	500	2

s		
empid	Sal	dept
1	100	5
2	200	2
3	300	5
4	400	2
5	500	2

Q1: ✓

Q2: {2, 3, 4, 5} ✗ [Replace inner query with 100, 300).

→ Non-correlated Sub Query. ✓

Select operation in SQL.

→ Projection opⁿ in RA except that SQL retains duplicates. ✓

Gate 2014

employee (empid, ename, edept) 1
customer

→ every sales rep id must be empid.

correlated Sub Query

employee		
eid	ename	edept
→ 1	a	X
2		
3		

customer		
Cust id	Name	sales rating
4		1 good
5		2 bad
6		3 avg
7		1 good
8		1 good

(2)

(1)

(d) all customers having a good Rating. ✓

Gate 2016 Q8.

Add of theatre with max capacity.

Cinema P1	
Add	capacity
a	100
b	200
c	300
d	400

P2	
Addr.	capacity
a	100
b	200
c	300
d	400

(a) d ⇒ Non-correlated query

(b) (i) {100, 200, 300, 400}

(c) ✓ a) 400 i.e. d

✗ b) {b, c, d}

c) > 400 ⇒ No tuple.

d) > 400 ⇒ No tuple. ✓

Gate 2016 (Imp Q)

'with' → naming SQ.

total	
name	capacity
Ajmer	20
Bikaner	40
Churu	30
Dungarg.	10

total-avg
capacity
25

{Bikaner, Churu}

* Gate 2012

P. Having clause should be applied only with Group by clause.

↳ ∵ need to filter groups.

→ All attributes must appear in Select clause if in Group by, else part in Aggregate funct^n

* RELATIONAL ALGEBRA GATE QS

Q Gate 98 Q

assume LHS is right

a) $\sigma_{C_1}(\sigma_{C_2}(R_1)) \longrightarrow \sigma_{C_2}(\sigma_{C_1}(R_1))$

Selection is commutative.

b) $\sigma_{C_1}(\Pi_{A_1}(R_1)) \longrightarrow \Pi_{A_1}(\sigma_{C_1}(R_1))$

c) $\sigma_{C_1}(R_1 \cup R_2) \longrightarrow$

R_1		R_2		$R_1 \cup R_2$	
A_1	A_2	A_1	A_2	A_1	A_2

d) $\Pi_{A_1}(\sigma_{C_1}(R_1)) \longrightarrow \sigma_{C_1}(\Pi_{A_1}(R_1))$

can contain A_1, A_2, A_3 anything.

R_1

A_1	A_2	A_3

A_1	A_2	A_3

Selection is applied if C_1 on $A_2 A_3$ X.

Assumpt^n: LHS is true.

Q $R \bowtie S$

<> → join condition [CP + Selection]

Q $\{t \mid t \in r \wedge t[A] = 10 \wedge t[B] = 20\}$ Gate 99

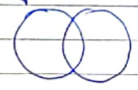
r

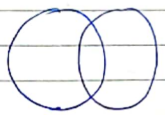
A	B	
10	20	

a) {10, 25}

b) $\sigma_{(A=10)}^{(r)} \cup \sigma_{(B=20)}^{(r)}$

c) $\sigma_{(A=10)}^{(r)} \cap \sigma_{(B=20)}^{(r)}$





Gate 2000

using RA operations.

a)
b) apply join again
c) Such of all

↳ Agg funct^n can be done using RA

Gate 2002

RA has same power of safe RC.

unsafe opr^n lead to infinite loop in RC.

Imp

Gate 2004

Student (name, sex, marks)

ρ → rename

≤m

Student

name	sex	marks
a	M	10
b	M	20
c	M	30
d	f	25
e	f	15
f	f	40

name	X	m
a	M	10
b	M	20
c	M	30
d	f	25
e	f	15
f	f	40

2nd half Query:

name	sex	marks	n	x	m
d	f	25	c	m	30
e	f	15	b	M	20
e	f	15	c	m	30

(def) − (d, e) = f

[a, b, c, f]

name of girls with marks lesser than atleast 1 male.

[Join]

Gate 2005

II laws:

R(A, B, C, D) — γ

R_1(A, B, C) R_2(A, D)

S(ABCD) — S

A	B	C	D
1	b_1	c_1	d_1
1	b_2	c_2	d_2
2	b_3	c_3	d_3
2	b_4	c_4	d_4

A	B	C
1	b_1	c_1
1	b_2	c_2
2	b_3	c_3
2	b_4	c_4

A	D
1	d_1
1	d_2
2	d_3
2	d_4

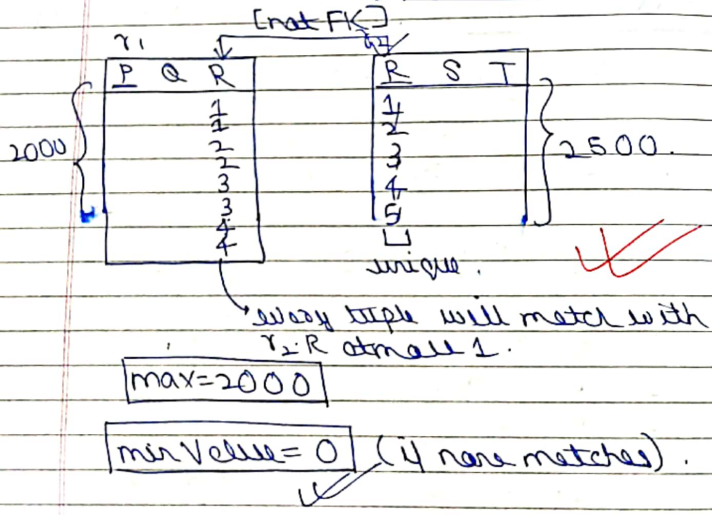
A	B	C	D
1	b_1	c_1	d_1
1	b_1	c_1	d_2
1	b_2	c_2	d_1
1	b_2	c_2	d_2

If common attribute 'a' is not a PK, we get Spurious tuples.

$\therefore$ $r \bowtie s$ → max tuples.

Gate IT 2006

$r_1 \rightarrow$ 2000T (P, Q, R)
$r_2 \rightarrow$ 2500T (R, S, T)
The max size of natural join.



max = 2000

minValue = 0 (if none matches).

Gate 2007

stud_info (studid, name, sex).

[All female id paired with all courses id]

eg

student_info

studid	sex
1	M
2	M
3	F
4	F
5	F

sid	cid
3	C_1
4	C_1
5	C_1
3	C_2
4	C_3
1	C_1
1	C_2
1	C_3

{C_1, C_2, C_3}

3
4 × C_1 C_2 C_3
5

$\Rightarrow$

sid	cid
3	C_1
3	C_2
3	C_3
4	C_1
4	C_2
4	C_3
5	C_1
5	C_2
5	C_3

if a girl hasn't taken

Final

C_3 not taken by 5

sid	cid
3	C_3
4	C_2
5	C_2
5	C_3

C_2, C_3

[if atleast one girl has not registered].

Gate 2008 ✱ → NJ

R(P, Q, R_1, R_2, R_3)
S(P, Q, S_1, S_2)

eg

R

P	Q	R_1	R_2	R_3
1	2	–	–	–
2	3	–	–	–

better

S

P	Q	S_1	S_2
1	2	–	–
2	1	–	–

a) 12 a b c ab. ⇒ 1.

b) {1,2} ✱ {1,2}. $\frac{1}{2}$ $\frac{1}{2}$

$\{\frac{1}{2}\}$.

c) P Q → {1}.
1 2

d) A − (A − B)

A B

$A \cap B$.

III & IV are same.

Gate 2012 on RA.

$(A \cup B) \bowtie_{Aid>40 \vee C.id<15} C$

A ∪ B.

Id	Name	Age
12	Arun	60
15	Shreya	24
99	Rohit	11
25	Hari	40
98	Rohit	20

C

Id	Phno
10	2200
99	2100

Aid	Cid
12	10
15	10
99	10
25	10
98	10
12	99
15	99
99	99
25	99
98	99

7 tuples

Take ∪ or else you will get 9.

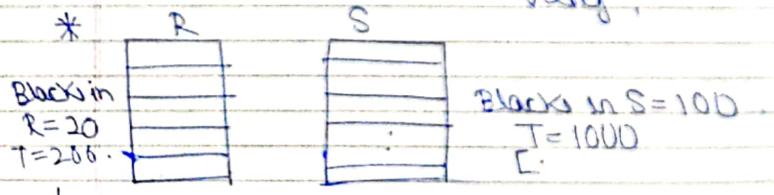
Gate 2014

r & s using nested loop method.


```
for (every tuple in R)
    for (every tuple in S)
    {
        // comparison
    }
```


```
for (every t in S)
    for (every t in R),
    {
    }
```

disk blocks might vary.



for each tuple in R, move all blocks of S.

100 * 100

[20 + 200 * 100] : Total disk mov

⇒ 20020

Now reverse:

100 + (1000 * 20) = 20100

(for each tuple in S)

[for every record, 1 block has to be moved]

r(R) shud be in outer loop for better performance.

Join Selection factor = o/p tuples out of original tuples

Gate 2014

eg)

dept	eid	deptname
10	1	
15	1	
20	2	
40	2	
45	3	
46	3	
50	4	
51	4	

empid	empName	age
1		20
2		30
3		40
4		50

Join all employees with his dependents whose age of dept is older than him

empid
2 — 1 20 1 10.
3 — 1 20 1 15.
4 — 40 2 2 30
45 3

(b)

∴ o/p = 1 [age greater than all its dependents]

Π_{empid} (employee ⋈ dependent) (empId = e.id) ∧ (empAge ≤ deptAge). a

empid whose age is less than atleast one of its dependents.

Full Natural Join = NJ + FOJ

R1

A	B
1	5
3	7

R2

A	C
1	7
4	9

A	B	C
1	5	7
3	7	N
4	N	9

(c) e, f, g but not a, b.

Gate 2007

{e.name | employee(e)} → e / x [using employee table 2 times]

∧ (∀x . . .) → ∀x(p → q)

p

(employee(x) ∧ x.supervisorN = e.name) → x.sex = 'm'

q

e

name	sex	superN
a	m	a
a	m	b
b	f	a

x

name	sex	Sup.No
a	m	a
a	m	b
	m	a
b	F	a

p is false.

All the names of employees who have more subordinates. (supervisors)

Q Gate 2006.

Q Gate 2008

$\forall t \in r(P(t))$ C

1. X
2. X
3. $\forall t \in r(p(t))$ ✓
4. $\exists t \in r(p(t))$

2. $\exists t \notin r(\neg p(t))$. X . → $\exists t \in r(p(t))$

* UnSafe Relational Calculus

a) $\neg\exists$ ✓ safe.
↳ there doesn't exist.

c) $\{t \mid \neg(t \in R_1)\}$

* Safe & Unsafe Query

$\{S \mid S \notin \text{Sailors}\}$ → unsafe Queries.

* Domain of Query:

↳ All tables used in query.

∴ Domain = Sailors

2. $\{T \mid \exists S \in \text{Sailors} \wedge \exists r \in \text{Reserves}\ (R.\text{sid} = S.\text{sid} \wedge R.\text{bid} = 103 \wedge T.\text{name} = S.\text{name})\}$

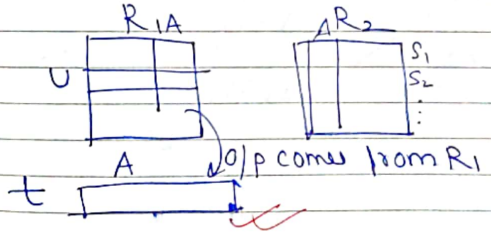
∴ Domain : S, R.

o/p ⊂ Domain of tables ✓ ∴ Safe.

$\{t \mid \exists U \in R_1 (t[A] = U[A]) \wedge \neg\exists S \in R_2 (t[A] = S[A])\}$

↓ (there exist no S belong to R_2 such that $t(A) = S(A)$)

$\{t \mid \exists U \in R_1 (t[A] = U[A]) \wedge \forall S \notin R_2 (t[A] \neq S[A])\}$



Q $\forall t \in \gamma(P(t))$

$\forall t \in$ Student (t pass Physics exam)

1. $\neg\exists t \in \gamma(P(t))$ ✗
   ↳ No Student Pass Physics exam.
2. $\exists t \notin \gamma(P(t))$
   ↳ there is a person not a student, but passed Phy.
3. $\neg\exists t \in \gamma(\neg P(t))$
   ↳ there is no student who failed Phy exam.

TRANSACTION MANAGEMENT

Q Gate 2012

- concurrent execution → interleaved only in conflict

T_1	T_2
read(P)	
read(Q)	
if P=0 then Q=Q+1	
write(Q)	
	read(Q)
	read(P)
	if Q=0 then P=P+1
	write(P)

∴ $T_1 \to T_2$ / $T_2 \to T_1$ (not possible).

Any non-Serial interleaving of T_1 & T_2 for concurrent execution leads to

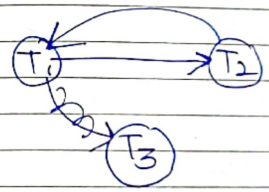
T_1: read(P)	T_1: read(P)
T_2: read(Q)	T_2: read(Q)
T_2: write(P)	T_1: write(Q)
T_1: write(Q)	T_2: write(P).
T_2: read(Q)	T_2: read(Q)
T_1: read(P)	T_1: read(P)
T_1: write(Q).	T_2: write(P)
T_2: write(P)	T_1: write(Q)

∴ Nothing is conflict Serializable. (B)

Q Gate 2014

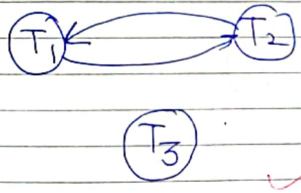
$r_i(x)$, $w_i(x)$

(A)

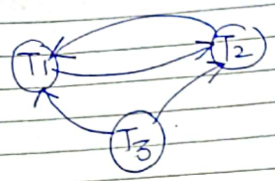


✗

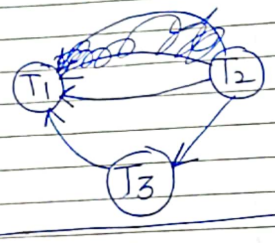
(B)



✗

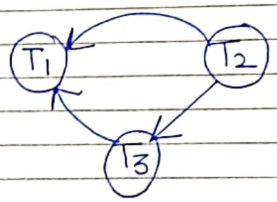
(C) 

X

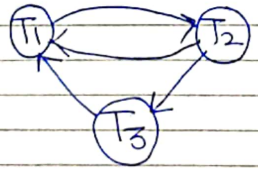
(D) 

(No cycle).
∴ ✓ conflict Serializable

Q Gate 14 Q.



∴ conflict Serializable ✓
S_1



S_2.
Not CS.
(A) ✓

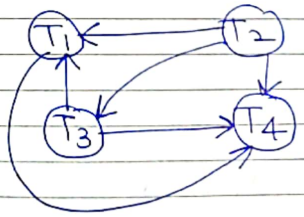
Q Gate 2014 Q

recoverable : see the reads.

T_1	T_2	T_3	T_4
	$r(x)$		
		$w(x)$ T_4 ✓	
		commit	
$w(x)$ T_4 ✓			
commit			
	$w(y)$ T_4 ✓		
	$r(z)$		
	commit		
		$r(x)$	
		$r(y)$	

↳ after write, we immediately commit, ∴ cascadeless also. [no dirty read] ✓

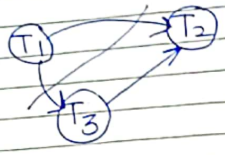
T_1 $w(x)$ → T_2 $r(x)$. (commits b4. T_1).



No cycle.
∴ CS & recoverable ✓

✓

Q Gate Q on CS Schedule



(A) $T_1 T_3 T_2$

Q Gate 2016

a_i ⟶ abort by T_i.

↳ if due to some problem, unav. of resources, rollback all changes.
[dont commit]

T_1	T_2
	$r(x)$
$r(x)$	
	$r(y)$ ⟶ read directly from DB.
$w(x)$	
$r(y)$	
	$w(x)$
abort	
	abort

· recoverable ⟶ write/read.

[No dirty read here]

∴ recoverable & cascadless
S does not have a cascading abort.

Strict.

T_2 writes data b4 t1 commits.

Q Gate 2015

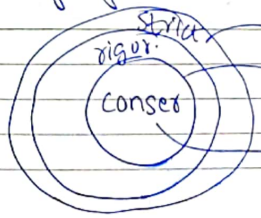
T_1	T_2
read(x)	r(y)
x = x − 50	y = y + 50
write(x)	w(y)

∴ (B) Consistency.

Q Strict 2Phase Lock

a) not strict ⟶ unlocking before commit
b) strict
c) strict ⟶ ∵ exclusive locks are unlocked after commit.

— Every rigorous is strict.



FILE STRUCTURES – Prev Years.

- Refer 6.16 from database Prevs Grp.
- B+ Tree is preferred to BST, since data transfer from disk is in BLOCKS. [& B+ tree can hold large cnt of keys in a block/node].
- B Tree Q. ⟶ 6.5
- → B+ Tree Insertion → 6.12, 6.13.
  6.15 Secondary index.

* View Serializability

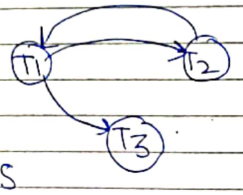
A Serial Schedule 'S' & non-serial schedule 'S' said to be view equivalent if:
- same trans. performing initial read in both. (for every data variable)
- same trans. performing final write in both.
- same trans. reading value written by other trans. in both.

eg1: T_1 T_2 T_3

R(A)
W(A)
W(A)
W(A).

View serializable?

≡



× [Not CS]

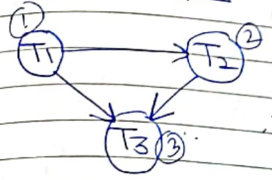
∴ Test VS

$W_2(A)$ ↗

Blind write: writing before reading.
[If no Blind write, & not CS, its VS]
Here, we cant comment.

Draw Polygraph:

initial read: T_1
final write: T_3



∴ VS ✓ [$T_1 T_2 T_3$]
(draw this take care of WR dep!).

eg2:

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

(Not CS).
∵ 1 blind write
(Test for VS).

initial read = T_1
final write = T_1

∴ Not VS ∵ cycle.

eg3:

T_1	T_2
R(A)	
W(A)	
	R(A)
	W(A)
R(B)	
W(B)	
	R(B)
	W(B).

CS ✓
∴ obviously VS ✓

* Polygraph Example:

$S = r_1(A)\ w_2(A)\ r_3(A)\ w_1(A)\ w_3(A)$.

T_1 T_2 T_3

$r(A)$ — $W(A)$ → $r(A)$

$W(A)$ — $W(A)$.

Init read = T_1, final write = T_3

$T_1\ T_2\ T_3$ → serial schedule. [view eq. to this].

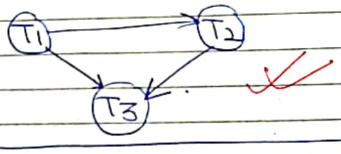
CS

T_1 ⇄ T_2 ✗

T_3

∵ Blind write. (Not check VS)

VS



✓

eg1 $S = T_1: r(x),\ T_2 = w(x),\ T_1 = w(x)$.

Imp

T_1	T_2	
$R(X)$		
	$W(X)$	→ blind write.
$W(X)$		
abort		→ ∴ abort. [na cascading abort].
commit		

→ delete T_1.

∴ [If 1 transaction → CS & VS both]

CS2

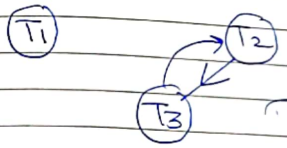
T_1	T_2	T_3	
$R(X)$			✗ init read: T_1, final W: T_1
	$R(Y)$		
$W(X)$			✓ init read: T_2
	$R(Y)$	$W(Y)$.	→ WR conflict $T_3 \to T_2$
		$R(Y)$	

∴ Not CS. (loop b/w T_3 & T_2).

⇒ Blind write: $W_3(Y)$; check for VS.

✗ init read: T_1,

Imp



→ Not VS

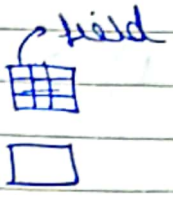
T_3 after T_2 definitely.

(T_1 can be anywhere, doesn't matter).

can get only Serial Schedule which is VE to given Schedule.

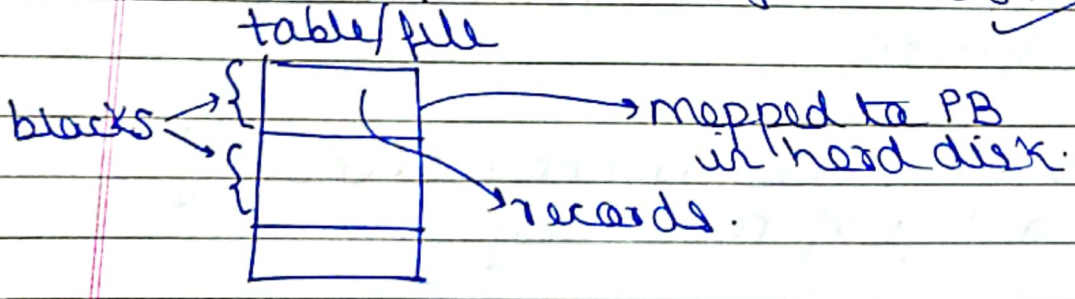
FILE ORGANIZATION

DB: collection of data (collection of files).



file
|
record
|
field

→ in OS: we want to access file from begin to end.
in dbms: we need 1 particular record.
∴ File org of dbms is diff from file System-OS.



avg no. of records per block = Blocking factor.

$$\frac{1024B}{4B}$$

↳ record size.

- Strategies for storing file of records into blocks.

1) Spanned Strategy → (Call)

R_4 → not stored completely.
B_1 fully utilized.

To access R_4 → 2 Blocks accessed. [B_1 & B_2].

- If Var Size records, use this.

eg: | size | content | → varsize.

but employee table: unspanned

2) Unspanned Strategy → (Call)

No record can be stored in more than 1 block.

- No. of block access to acc. a record ↓.
- But, wastage of memory.

Suitable for fixed length record.

* Organization of records in file

for storing data in HD, order must.

1) Ordered file organization → (Call)

emp

SSN
1
2
3
.
.
7

records are ordered based on a field (SSN)

if empT → 100B.

100 x 10 ← records.

B1	B-2	B-3	...	B-100
1–10	11–20	21–30		990–1000

record no 25. → apply B.S (Bin Search)

- if search based on key on which ordering is done → efficient.

Insertion is expensive

if 22 record put up. lot of move

2) unordered → (Coll)

All records in file are inserted wherever the place is available at the end of file.
Search: LS. (Search every Block).
Search is inefficient.

* Intro to Indexing

- Size of each record big.
-
BS ⟶ $\log_2 1000$ → 10 Searches / block access to access a record.
∵ ordered based on Key.

Maintain index

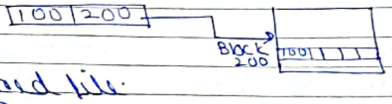
index file small,
[Key, Block Pointer].

also divided in blocks (8).
∴ $\log_2 8 = 3$ block access
∴ $\log_2 8 + 1 \Rightarrow 4$

if file is unordered, index file can be ordered
But if sorted,
(1 index record for a block)

* Structure of index files

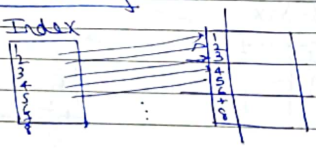
[Key | Block Pointer]
[Block Add where Key is available.



- Index = ordered file.
- Searching: BS.
- To access a record using index.
avg no. of block access.
$\log_2 Bi + 1$.
↳ data block access.
↳ index block access.
- Index can be created on any field of relation; ordered / unordered file.
[PK, non key, CK].

* Classification of Indexing

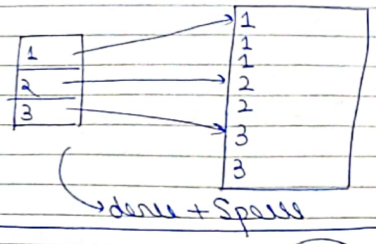
1) Dense index:



for every search key value: index entry.

2) Sparse index: index entry created only for some records.

*



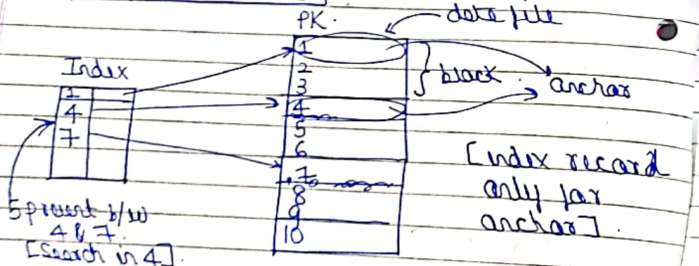
* Types of indexing → (Cell)

if 1 index file : Single level index.
indexing on index file: multi level

* Primary Index (Primary Key + ordered)

DF ordered based on PK. (Cell)

PK	Block Pointer



[index record only for anchor].

→ No index record for 2 & 3.
Sparse indexing

- No. of index entries = No. of data Blocks
- Avg no. of block Access.

⇒ B_i blocks in index.

$\therefore \boxed{\log_2 B_i + 1}$ → to get actual data

↳ after finding req block in index.

* Example on Primary index

Q → Cell. Unspanned.

records are fixed.
Find no. of block Accesses with / without index.

Block Size = 1024B.
Records = 100B.
Records / block = $\frac{1024}{100} \Rightarrow 10.24$ → can't put 11 records. ∴ unspanned. ⇒ 0.24 waste

$\therefore \boxed{10}$

∴ No. of data blocks = $\frac{30,000}{10}$ = 3000 Blocks needed to hold all records.

If we have ordered file (based on PK)

Search based on PK, then how many block access → $\lceil \log_2 3000 \rceil$

1) Without index ⇒ 11. ⇒ 12

2) With Primary index

index → 15B.

index record = 15 B.

index record / block = $\frac{1024}{15}$ = 68. ⇒ 68 (floor)

No. of index records = 3000.

∴ [No. of blocks required in index = $\frac{3000}{68}$ = 44. ⇒ 45]

$[\log_2 45 + 1]$ ⇒ 8. 6 + 1 = 7

* Clustering index → (all) → retrieve all records with accurate of Key.

- ordered on non-key field. [repeated]

[for every distinct Key value → 1 index record]

↳ Space & time both

No. of block Access ≥ $\log_2 B_i + 1$

B_i → no. of blocks in index file.

* Secondary index

if Primary index → Empno. & SQL query based on ena → OK.

But if we get query on ename, not used.

∴ Maintain Separate index.

index → ordered.

↳ but on unordered data file.

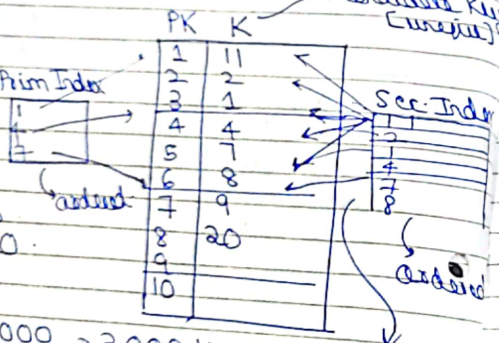
∴ BS on index file.

→ index created for every record in file. [Dense] (Not Sparse).

→ [No. of index entries = no. of records].

* Example on Secondary Index

30,000 records on a disk
Block Size = 1024 B
record size = 100 B
→ ordered file based on PK
Primary index already exists
Secondary index on Key (∵ no repeat)
↳ on unordered file
[from SI]



Block Size = 1024
record size = 100 B
records/block = 10

no. of blocks required by ordered file $\Rightarrow \frac{30,000}{10} \Rightarrow 3000$ blocks

for every record: 1 entry ∴ dense

∵ unordered based on key
⇒ ~~30000 Block A~~
⇒ $\left(\frac{1+3000}{2}\right) = 1500$ Block Access

(without index)

Size of index record = 9 + 6 = 15 B (Key + data pointer)
No. of index records in Secondary index = 30,000

No. of index records/block = $\frac{1024}{15} \Rightarrow 68$

∴ No. of blocks required by the index file = $\frac{30,000}{68} \Rightarrow \frac{442}{}$ [441.—]

$\therefore \left[\log_2 442 + 1\right]^{8.—} \Rightarrow 9+1 \Rightarrow (10)$

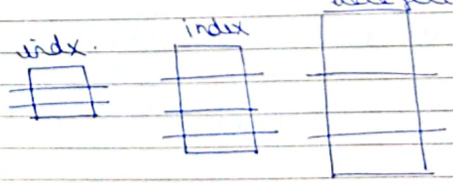
Q Gate Q on 13.08

An index is clustered if:
data records of file are arranged in the same order as the data entries of index.

Q clustering → non-key & ordering

* B-Trees 1

↳ Multilevel indexing made (Generalized)



- B & B+ dynamic → size inc/dec.
  (general of multi level indexing).

(ii) node Pointer / block Pointer [add of block]
(iii) record Pointer
[Points to the record we are searching for]

BP = 6B (identify which block)
if 1024 records in block = 10B
(16B)

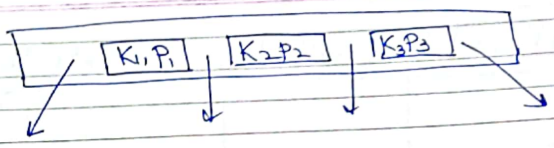
* Key + data Pointer → block/record

Key = value on which we search.

Root can have children b/w 2 & p.
order.

* B-trees 2

Internal nodes can have children b/w $\lceil p/2 \rceil$ and p.
- can have keys b/w $\lceil \frac{p}{2} \rceil - 1$ and $p-1$



children = KP + 1

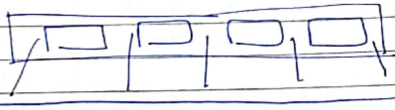
* Leaf Nodes

Keys b/w $\lceil p/2 \rceil - 1$ and $p-1$ → same as internal

→ dist. from root to any leaf is same.

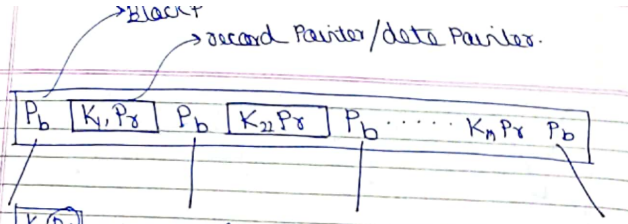
p → order → how to get p?

eg
in root → min children = 2.
→ max children = 5.



* Example 1 on B-Trees

Key Size = 10B
Block Size = 512B.
data Pointer = 8B
Block Pointer = 5B.



if order = n (Assume)

$n P_b + (n-1)(K + P_r) \leq$ Block Size [512].

(P_b: no. of Block pointer; K: Size of Key)

$\therefore n5 + (n-1)[10+8] \leq 512$

$\Rightarrow 5n + 18n - 18 \leq 512$

$\Rightarrow 23n \leq 530$

$\Rightarrow n \leq \frac{530}{23} \approx 23.\_\_ = 23$

$\therefore$ Max children = 23 (p) ✓

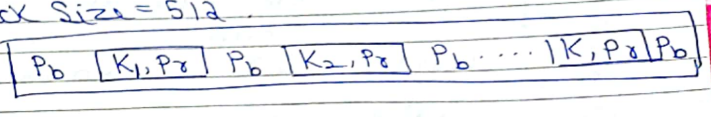
	children	Keys
Root →	2, 23	1, 22
internal →	$\lceil 23/2 \rceil$, 23	11, 22 ✓
	12, 23	

* Example 2 on B- Trees

if searching anything less than K_1 go to P_b (add. of node in next level).

15

Key = 9B.
P_r = 7B → associated with Key.
P_B = 6B.
Block Size = 512.



Let order be n.

$n * P_b + (n-1)[K + P_r] \leq 512$ (512: Size of node)

$\Rightarrow 6n + (n-1)(16) \leq 512$

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

$22n \leq 528$.

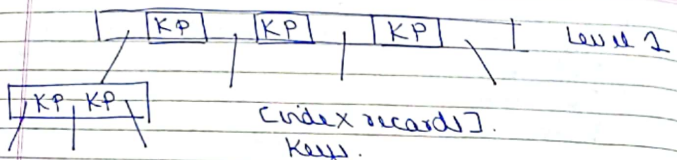
$n \leq \frac{528}{22} = 24$.

$\therefore$ 24 [Max no. of children]
= Mid. ↳ ht of tree will be less.

root: 2, 24.
internal: 12, 24. ✓

* Example 3 on B Trees

Order of B-tree = 23.
4-level.



Level 1

[index records]
Keys.

Level 1 : 22.

Level 2 : 23 * 22. → max.

Level 3 : (23)(23)(22)

Level 4 : (23)(23)(23)(22)
nodes in L3 — children.

* Underflow and overflow in B-Trees

if order = p → max no. of children.
Keys exceed p-1 [max]
{root / internal / leaf}.
↳ overflow.

eg: insert 9. (b/w 8 & 10).
but already max keys. [p-1].
↳ Overflow

* Underflow → delete a node
[merging req].

Root : min keys 1
internal : min keys = $\lceil p/2 \rceil - 1$

if [no. of search key values < $\lceil p/2 \rceil - 1$] (Underflow)

node has what it is actually required.

if p=5 → 2 (min).

delete 10.
↳ became 1 < 2. [Underflow].

* B-Tree Search

- Search index record in B-Tree

→ We have p choices to make at each node [p way search]

eg: Search record with Key Value 17.

↳ first get into root.

if match: stop at root

Adv: need not go till last node.

17 < 25.

↳ go left.

[Keep searching in linear fashion].

till 18,.. go left.

↳ we get 17 at Level 3.

∴ All index records are distributed over B-tree.

(No of comparison very less).

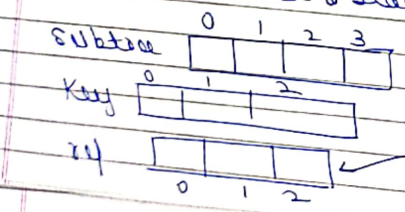
* B-tree Searching Algorithm

- search the node, till we find a Key greater than the Key we are searching for & go left.

if equal to: return it.

Stop if: • find exact Key.

• till end & Stop.



tk → Target Key.

tk < node.Key[i]:

break;

• Algo → Call.

as, order ↑, Time ↓ → Goto

Time complexity: $\log_p n$ → depth of tree.

p → order of B-tree

n → no. of Search Key.

* Insertion → Steps (Call)

first Search the position.

- Split if no Space.

* Eg 1 on B-tree Insertion

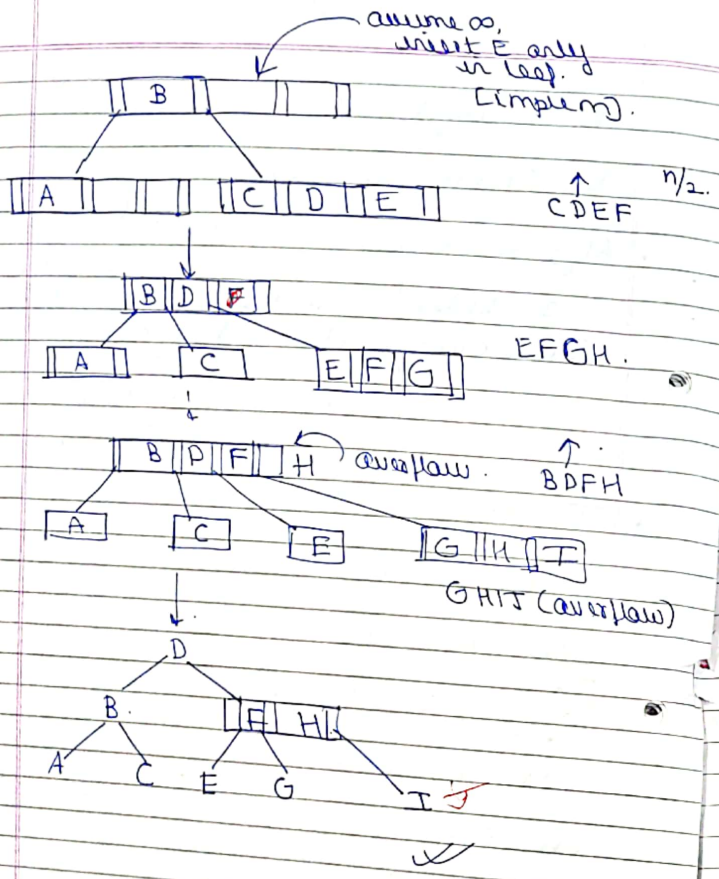
order of tree = 4. → max = p − 1

→ min = $\lceil \frac{p}{2} \rceil - 1$

insert A, B, C D E F G H I J.

if D: ov

ABCD.



Example 2 on B-Tree insertion

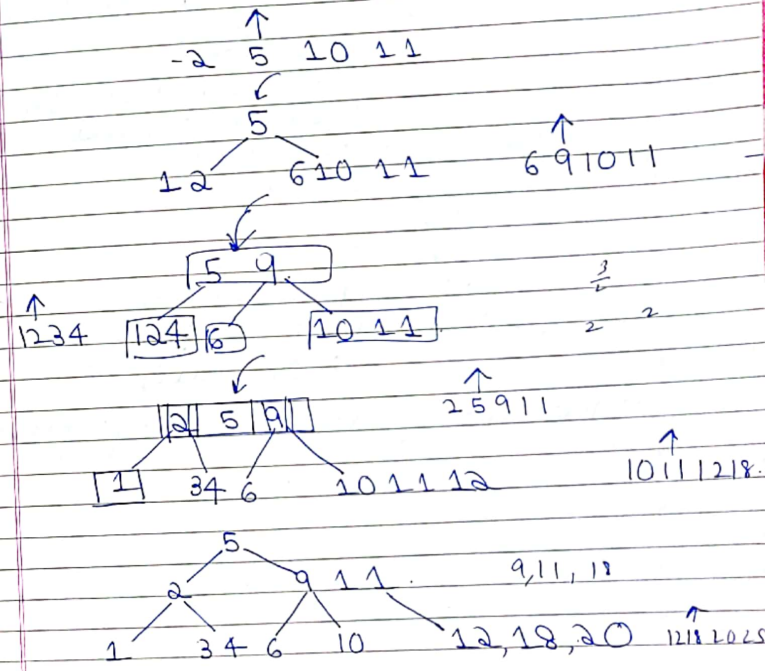
concerned about overflow.

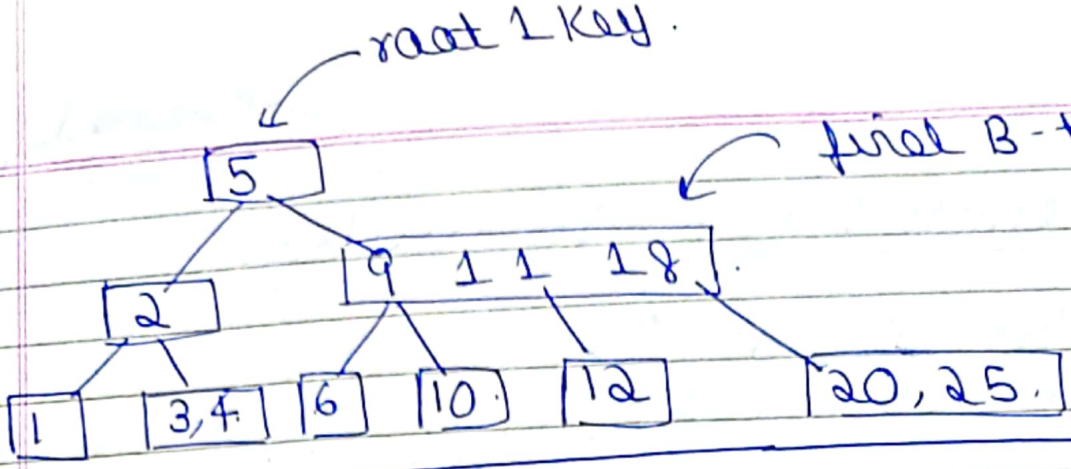
B-tree of order 4.

Key: 2, 5, 10, 11, 1, 6, 9, 4, 3, 12, 18, 20, 25.

Root
min = 1 key
max = 3 keys

Internal.
min = 2 ~~keys~~ - 1 = 1.
max = 3 keys





we split the nodes.

if Order = "p".

max = 'p+1' + 1 = p keys (overflow).

split.

median is promoted up.

[p-1 keys adjusted over 2 nodes].

p=odd		p=even	
$\frac{p-1}{2}$	$\frac{p-1}{2}$	$\lfloor \frac{p-1}{2} \rfloor$	$\lfloor \frac{p-1}{2} \rfloor + 1$.

All satisfy min criteria.

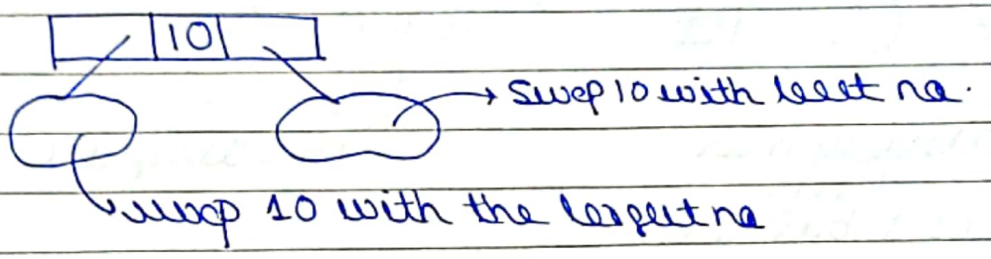
∴ Dont worry abt underflow.

[in Insertion].

2) When inserting elem, how many splits?

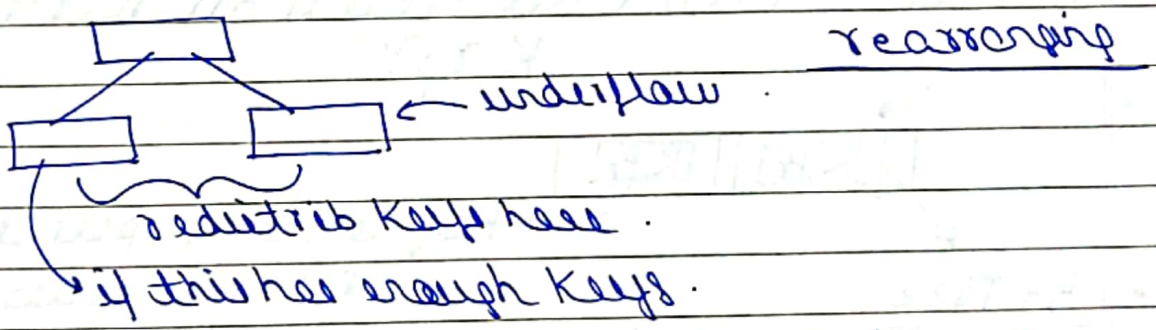
* Deletion from a B-Tree 1 → concern abt underflow

- deletion should



- $\lceil \frac{P}{2} \rceil - 1$ → min keys any leaf can have.

- If underflow: consider 2 immediate Siblings.



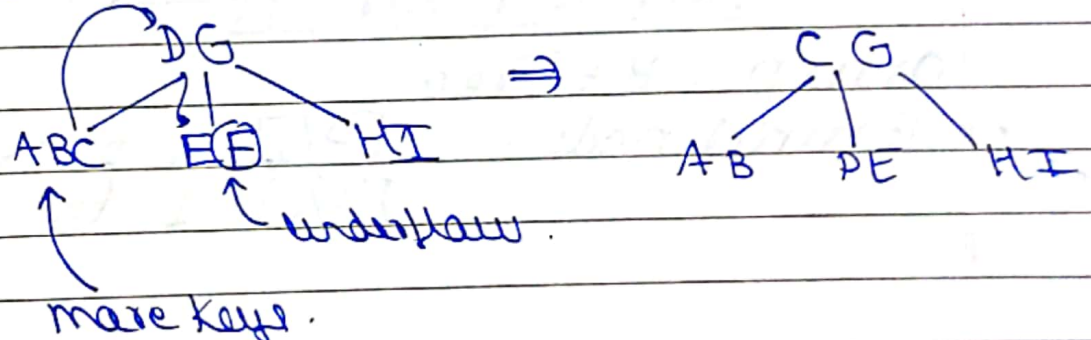
rearranging

→ If both immediate Sib. have exactly min no of key, then merge. coalescing.

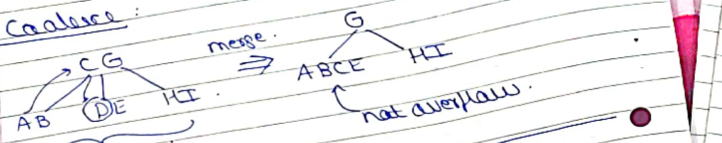
→ If parent key → underflow: prop. upwards

* ~~Del~~ from B-tree 2

Borrow: 5 order tree → $\lceil \frac{5}{2} \rceil - 1 = 2$

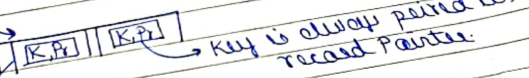


Coalesce:

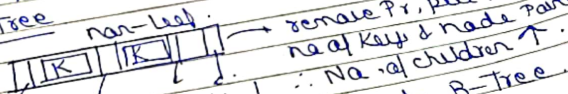


* B+ TREES

- B Tree → distr index records all over the tree. (K, rp)



- B+Tree



⇒ Leaf contains K Pr, similar to B-Tree (no spec for record Pointer).

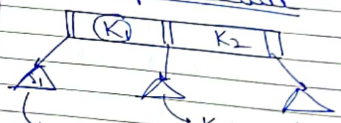
- Order p :: B + Tree
- Internal node: $\lceil p/2 \rceil$ to p children. $\lceil p/2 \rceil - 1$ to p−1 keys.

25

O = 3 (order of non leaf).

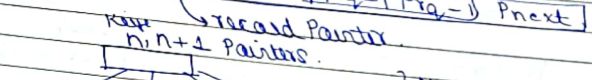
No. of children = 3
No. of Keys = 2
} max

* Non-leaf Structure



Key Val in $S_1 < K_1$
Key value in $S_2 \geq K_1 < K_2$

* Leaf Node



n keys,
n + 1
p
NP.

Order of Non-Leaf > Order of Leaf.
∴ rp more size.

Order of non-leaf : max no of children it can have
Order of leaf : max no of K-V pairs present inside the leaf.

If both the nodes pointers same order.

Example 1 on B+ Trees

Search Key = 9 B
Block Size B = 512 B
Block P = 6 B
record P = 7 B

Internal node (root/non leaf)

| P_b | K_1 | P_{b_2} | K_2 | P_{b_3} | K_3 | ..... | K_{n-1} | P_n |

1 extra than Key.

$n(P_b) + (n-1)K \leq 512 B$
$6n + (n-1)9 \leq 512$
$\Rightarrow 15n - 9 \leq 512$
$\therefore 15n \leq 521$
$n \leq \frac{521}{15} \leq 34.5$

$\therefore$ n = 34 ← Order of Non-Leaf. i.e No. of Block P, it has.

Leaf Node

| K_1 | P_{r_1} | K_2 | P_{r_2} | K_3 | P_{r_3} | .... | K_m | P_{r_m} | P_b |

$m[K + P_r] + P_b \leq 512$
$m[9 + 7] + 6 \leq 512$
$16m \leq 506$
$m \leq \frac{506}{16} = 31.$
$\therefore$ m = 31

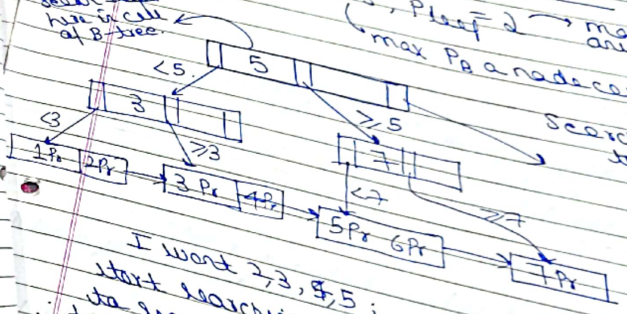
Order of Leaf = 31. (max KV pairs it has).
If any Block Pointer in Leaf
Order will be 1 less as same.

* Searching B+ Tree Algorithm

B+ Tree with P=3, P_{leaf} = 2 → max Key V pairs any leaf can have
(max P_B a node can have.

Search always term in leaf.

Search stops here in all of B-tree.



I want 3, 3, 5;
start searching with 2, to find 3, no need to search from root.
to efficiently implement range query

Breadth ↑ ; Search time ↓ ∴ depth ↓.

Time Complexity. Ass O(L) = O(N.L) = same
→ order, value

$\log_p^n$ → much higher th

Q Gate Q-2015 (B+ Tree)

≤ 7 and < 15.

- records pointers thru, but not shown
- Here, doubly linked list.

[7, 15)

<9 9

5 >=5 ——→ travelling this way.

5 7

∴ Min nodes fetched/visited = 5 ✓

Q Gate IT-2015

Table has 4 levels. (including root)

Assume all nodes are full (p-1) Keys.

Na of nodes newly created

5 Nodes are newly created

∴ n levels → n+ Na

[P/2]-1

Q Gate 2015 Q [B+]

Key = 12 B.

Block size = 1024 B

PR = 10 B

Block Pointer = 8

Non-leaf.

$P_B K_1 P_B K_2 \ldots\ldots K_n P_{n+1}$

$n \times K + (n+1) P_B \leq 1024$.

$\Rightarrow 12n + (n+1)8 \leq 1024$.

$\Rightarrow 12n + 8n + 8 \leq 1024$

$20n + 8 \leq 1024$

$20n \leq 1016$

$n \leq \frac{1016}{20}$

∴ n = 50 (Max keys).

* Gate 2003

2-3-4. (2 or 3 or 4) [Children]

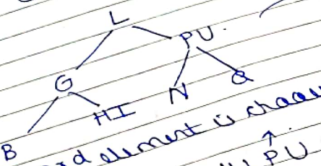
min Key = 1

max Key = 3

Lexicographical order used

BGHI ↑ ∵ win, either G/H can be chosen

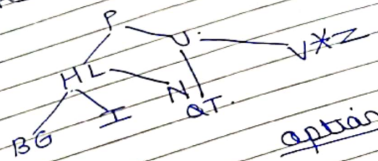
GLPU ↑ Key → [1, 3]



→ na option

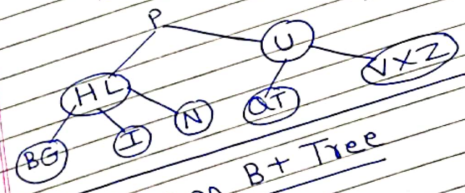
Here, 3rd element is chosen in middle.

BGHI ↑ HLPU ↑



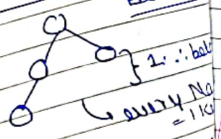
option matched

Q2,



* Gate 2016 on B+ Tree

→ balanced



Binary!

1. ∴ bal

query No = 110

nd are all equ

$\lceil \frac{p}{2} \rceil - 1$

-1 12 2025. 35

31

B & B+ Tree are perfectly Balanced.

* Gate 2010 on B+ Trees

① max Keys = 5. → Here data Pointers used in leaves also.

$n-1=5, \quad n=6$

② $\lceil \frac{6}{2} \rceil - 1 = 2$

if order = p, max keys = p−1 ∴ p=6

min key = $\lceil \frac{p}{2} \rceil - 1 = 3-1 = 2$ (b).

(↳ in non-root Node. root=1.

* Gate 2007

O(leaf) = max no. of K-V pairs. ↳ record Pointers.

P_r : 7 B.

Key : 9 B.

P_b : 16.

$m(7+9)+16 \le 1024.$

16 | n

$16m \le 1024 - 16 = 1008.$

$m \le \frac{1008}{16} \Rightarrow 63.$

∴ $\boxed{m=63}$

32.

* Gate 2008 on B-Tree

Order = 4 ; 10 successive insertion.

(max = 3)

max no. of node splitting ?

1 2 3

1) 1 2 3

2) 1 2 3 4 5

3) 1 2 3 4 5 6 7

4) 1 2 3 4 5 6 7 8 9

Max Split = 5

* B+ Tree Insertion

→ insertion starts from leaf.

(Key, record Pointer).

• index records only in leaf nodes.

• non-leaf : just insert key.

1 2 3 4

2 3 4 5 6

2 4 6 7 8 9

2 4 6 8 9 10

33

Leaf

Order 5 [B+ Tree]

max keys at any node = 4.

7 9 13 15

7 8 9 13 15

[Split]

even after splitting 9 should still be present

copy the smallest key value of 2nd node in parent.

Overflow : when no. of search key value exceed 'p-1'.

eg: order = 5

order is same for leaf & non-leaf = practical.

Overflow in Parent.

10 20 30 40

non-leaf

20

10 15 30 40

10 15 20 30 40

$\frac{n-1}{2}$

[Initial info → leaf].

* Example on B+ Tree insertion. → Worry about overflow

P=4 KC1,4,7,10,17,21,31,25,19,20, 28,42).

↳ order of leaf & Non-leaf both.

Max Key=3

Leaf |<K,P0> <K P1> .... PB|

↳ we need to insert Key + Pr.

Leaf: 1 | 4 | 7 | Pr

(14 7 10). Split.

⌈(n-1)/2⌉ = 2.

7 10 17 21

7 17

(1 4) → (7,10) → (17 21)

17,21 | 25, 31

17 19 21

17 19 | 20

7 17 | 20

7 | 17 20 25. 35

[Like B Tree]

20

7 17 X

25.

17

7 20 25.

1,4 7,10 (17,19) 20,21 25,28,31.

28.

17

7 20 25 31.

14 7,10 17,19 25 28 31 42.

20,21

42.

25 28 | 31 42

* Diff b/w B/B+ Trees & BST.

→ grow in breadth.

- less height comp to BST.
- Search time ↓

ht of B > ht of B+ tree

[Key-Pr]

[only Key: internal node)

more children.

B+ tree: Range based queries easily implemented.

Binary Tree → not preferred in dbms.

* B+ Tree Deletion — (call)

- entry in the leaves, & key in non-leaf also
B-Tree: key will appear only in 1 Node.

→ Check underflow.

i) L contains at least $\lceil \frac{p}{2} \rceil - 1$ entries, done!

ii) underflow, $L = \lceil \frac{p}{2} \rceil - 2$ entries.

iii) any of siblings, have keys greater than min. you can borrow.

→ iv) redist fail, merge ✓

If merge ✓, corresponding entry must be deleted.

replace with inorder successor.

* EXAMPLE 1 on B+ Tree Deletion

37

$5 \rightarrow \lceil \frac{5}{2} \rceil - 1 = \underset{\text{root} \rightarrow 1}{2}$ (min).

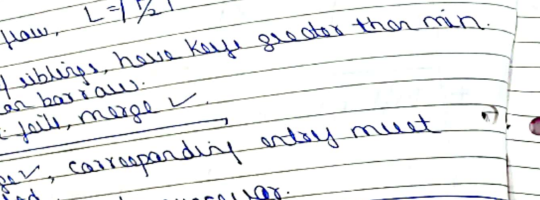
Delete 19, 20

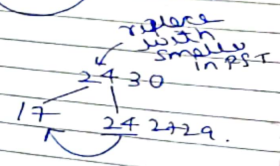
↳ find 19.

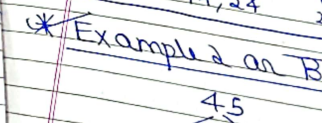
Delete 19 → 19 not present in non leaf. → No underflow also.

Delete 22

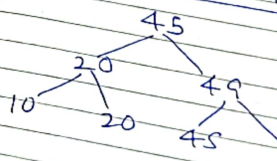
- Search 22
delete 22 → underflow (Call)







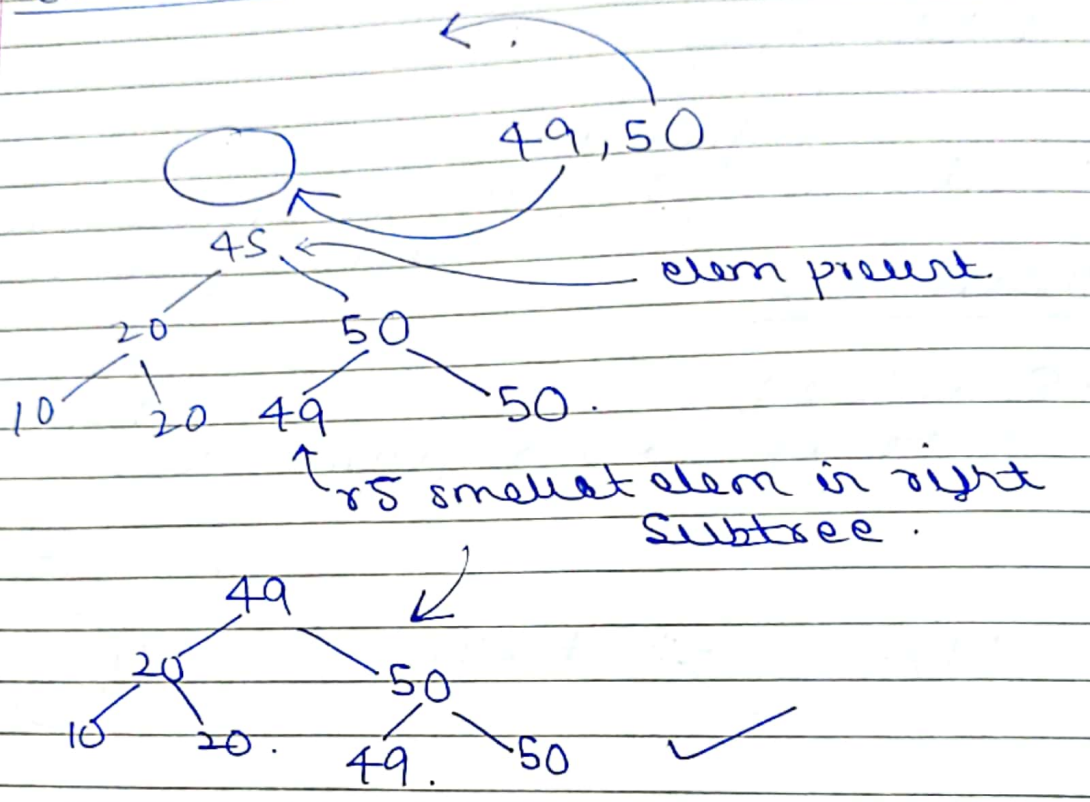
* Example 2 on B+ Tree deletion



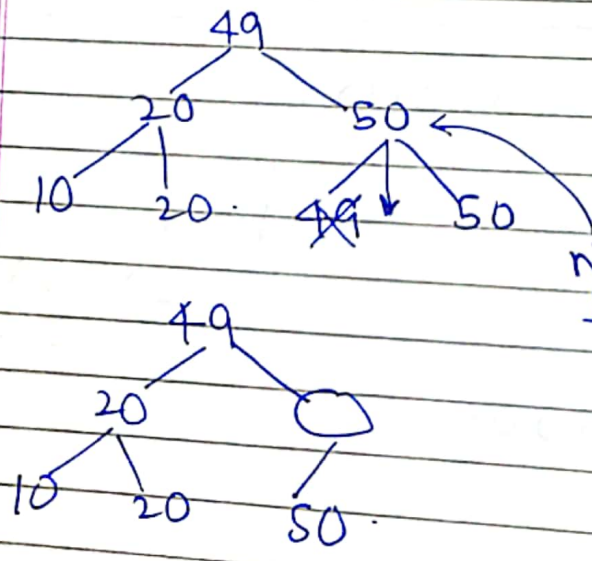
$\lceil \frac{3}{2} \rceil - 1$

(1) atleast 1 key.

delete 45.



Example 3 on B+ Tree Deletion



delete 49.

∵ no sibling.
can't borrow.
∴ Merge.