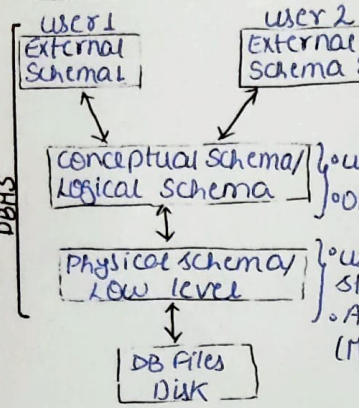
1) RDBMS Guidelines:
- Data must be in tabular format.
- No two rows of DB table must be same.
- Every attribute must be atomic (single valued)

2) Relational Schema: defination of table
3) Relational Instance: record set of table

⇒ Levels of Abstraction in DBMS:-



- Also called View level & High level.
- user interface provided
- subset of conceptual schema
- Data access to user based on role

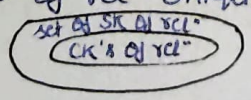
- used to represent what data in DB.
- only table name, attributes, key constraints

- used to represent HOW data is stored in DB files.
- All storage info maintained (Metadata or Data Dictionary)

1) Different levels of abstraction used to provide data independency i.e. can change low level without effecting high level.
2) Physical data independence: can change physical storage which may not effect conceptual details.
3) Conceptual data independence: can change conceptual details which may not effect view level.

1) Candidate Key:-
- Minimal attr set which can differentiate records of relation uniquely.
- Relation can have many candidate keys.

2) 1 CK is choosen as primary key, remaining CK's are called Alternate keys.
3) PK don't allow NULL value, whereas Alternate keys field value may be NULL.
4) Prime attributes: attr. which belongs to some CK's of rel".
Ex: R(A, B, C, D) Prime attr = {A, C, D}
Non-Prime attr = {B}
5) Super Key: set of attr (may not minimal) which can differentiate records of rel" uniquely.
6) Every CK is SK.
Not every SK is CK.



7) candidate key: Minimal SK.
8) Foreign Key: (Referencial key)
- FK is defined over 2 rel" (Referenced relation, Referencing rel")
- FK is set of attributes references to PK/AK of same rel" or some other relation.

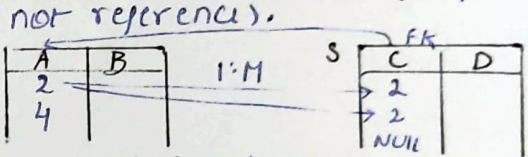
(PK/AK) Y ← X (FK)
(Referenced attr) (Referencing attr.)

9) FK may have duplicate or NULL values
10) Referential Key Integrity constraints:

Referenced Rel"	Referencing Rel"
• Insertion: NO violation • Deletion: May cause prblm. Possible sol": i) On delete NO action - if referenced, don't allow to delete. ii) On delete cascade - force referencing value to delete. iii) On delete set referencing value to NULL. • Updation: May cause violation. Some sol" as above.	• Insertion: May cause violation. If violation occurs, don't allow insertion. • Deletion: NO violation • Updation: May cause violation. If violation occurs, don't allow.

11) Referencing rel" records whose FK field value are NULL, are out of referential (do not reference).

12)

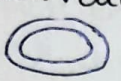
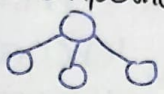


- Each record of referenced rel" relates to many (0 or more) records.
- Each record of referencing rel" relates to atmost one (0 or 1) record.

13) R(A B C) S(C D E) NOT allowed
R(A B C) S(D E) Allowed

Entity Relationship Model (ER)

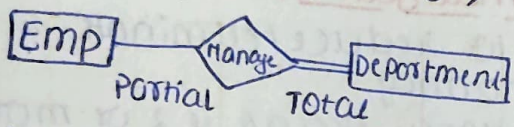
1) High level design of Database
2) Main components :- Attributes, entity set, Relationship set.
3) Entity set  (weak)  (strong)
4) Attributes 

Key  Derived  Multivalued  composite 

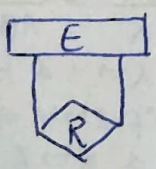
5) Relationship set 
6) Generally, verb is the relationship
7) Rel" may have attributes.

* Participation :

8) Total participation : all entities participate in relation (atleast+1) (=)
9) Partial participation : atleast 1 entity does not participate. (-)

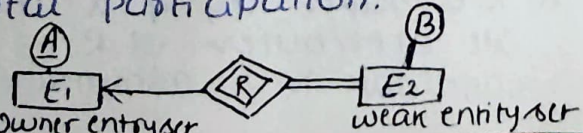


10) Weak entity set : no primary key 
11) Strong entity set : have PK. 
12) Weak Rel" set : exist blw strong & weak entity set. 
13) Strong rel" set : blw 2 strong entity set
14) Rel" is subset of cartesian product of A & B.
15) Degree of Rel" set = NO. of entity sets participating in a rel" set.
16) Self Referential Rel" Set (Unary Rel"): Entities of entity set E are related to some other entity of some entity set.



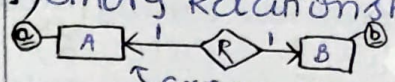
} Recursive entry set
} self Referential Rel" set

17) For each weak entity, there must be owner entity set which is strong.
18) Participation of weak entity set must be total participation.

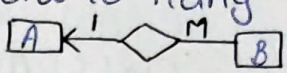


* Mapping cardinality :

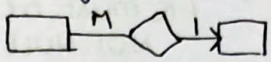
1) Binary Relationship (One : One)



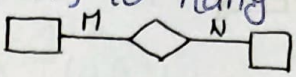
↑ arrow signifies cardinality 1.
CK of Rel" set = {A, B}

2) One to Many 

CK of Rel" set = {B}

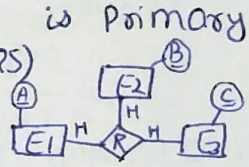
3) Many to one 

CK of Rel" set = {A}

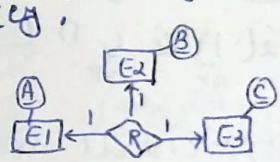
4) Many to Many 

CK of Rel" set = {A, B}

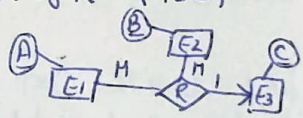
19) min cardinality = 0 ⇒ Partial participation
min card. = 1 ⇒ Total participation
20) Cardinality ratio : Max no. of Rel" in which an entity can participate
Participation : Min no of Rel" in which an entity can participate
21) Relation → Table
Tuple / Record → Row
Attribute / Field → Column
Relation schema → Table Heading
22) Degree of rel" : NO. of columns
23) PK can't be NULL or duplicate
24) Main integrity constraints of Rel" set is foreign key & of entity set is Primary key.
25)



CK of R = {A, B, C}



CK of R = {A, B, C}



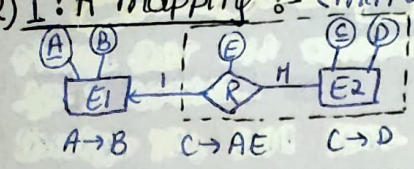
CK of R = {A, B}

NOTE : CK of R is combination of many sides key attributes.

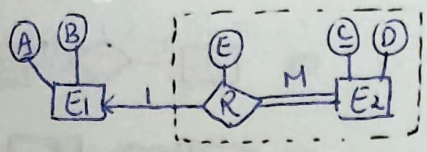
** Design Min number of RDBMS tables possible for given ER diagram.

(1) Partial participation on both side of binary rel"
- 1 to Many, Many to 1 } Merge rel" set towards many side (so, 2 table)
- 1 to 1 : Merge rel" set any 1 side so 2 relational table
- Many to Many : 3 tables, separate for all.

2) 1 : M mapping :- (min 2 table & 1 Foreign key)

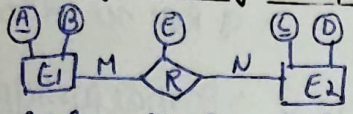


E1(A B) A→B
E2(C D E A) C→DEA
FK



E1(A B) A→B
E2(C D A E)
FK must be NOT NULL

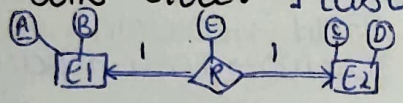
3) Many - Many mapping : min 3 tables & 2 FK



E1(A B) A→B
E2(C D)
C→D
R(A C E)
AC→E

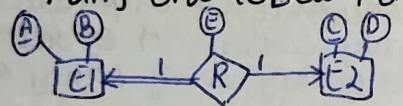
4) 1:1 mapping with partial participation at both ends : min 2 table & 1 FK
Merge Rel" table any side.

5) 1:1 mapping with total participation at both side : 1 table, NO FK



Alternative Key
NOT NULL PK
E1RE2(A B C D E)
{A, C} are CK

6) 1:1 , any one total participation



min 1 table, NO FK
no. of ele in E1 ≈ E2

E1RE2(A B C D E)
AK PK
may have NULL

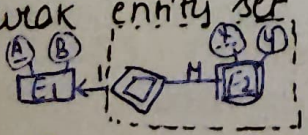
7) Self referential many to many, maintain separate table.
(min 2 tabl, 2 fk)

8) A strong entity set with any no. of composite attributes ⇒ 1 table

9) A strong entity set with any no of multivalued attributes ⇒ 2 table
1 contain simple attributes with PK
Other " all multivalued attri with PK

10) Rel" set requires 1 table having PK's of participating entities and its own attributes.

11) Weak entity set



E1(A B C)
E2(A X Y)

NOTE: (X) is partial key of weak entity set.

12) Multivalued & weak entity sets are allowed in ER diagram but not in RDBMS.

13) Self referential 1:1 or 1:M, min 1 table & 1 Foreign key.

14) Foreign key may have NULL or duplicate values.

15) Specialization - is a relation ▽
Generalization - has a relation

16) Candidate key - Minimal super key

17) Every candidate key is super key.

18) If there r N attributes with 1 CK
⇒ No. of possible super keys = 2^{N-1}

19) If CK's = {a1, a2}
⇒ No. of possible SK's = $2^{N-1} + 2^{N-1} - 2^{N-2}$

20) If CK's = {a1 a2, a3 a1}
No. of SK's = $2^{N-2} + 2^{N-2} - 2^{N-3}$

Normalization :

1) Used to reduce/eliminate redundancy

2) Redundancy occurs if 2 or more independent relations combined.

3) Redundancy results in anomalies - insertion, deletion, updation.

4) Functional dependency : X→Y
X can uniquely identify Y. ie
if $t_1.X = t_2.X$ then $t_1.Y = t_2.Y$.

(i) Trivial FD : X→Y is trivial if X ⊇ Y
Sid → Sid
Sid Sname → Sname

(ii) Non Trivial FD : if no common attribute in X and Y

(iii) Semi Non Trivial FD : combination of trivial & non trivial FD
Sid → Sid Sname & (Sid → Sid, Sid → Sname)

5) Armstrong Ruler over FD's :
- Reflexive : X→X , AB→A
- Transitive : if X→Y & Y→Z ⇒ X→Z
- Augmentation : if X→Y ⇒ XZ→YZ
- Union : if X→Y & X→Z ⇒ X→YZ
- Split : if X→YZ ⇒ X→Y & X→Z
- Composition : if A→B & C→D ⇒ AC→BD

6) X is super key if X^+ determines all attributes of R.
- candidate key : minimal SK.

7) If R(ABC) has no non trivial attr CK = {ABC}

8) If any attribute is not present in right side of any FD's then it must be present in all CK's.

9) TO PROVE: $X \to Y$ is member of FD set (F), iff X^+ determines Y in FD set (F)

10) TO PROVE: F & G FD sets r logically equal iff
- F covers G → All FD of G set must be derived from F (A)
- G covers F → All FD of F set must be derived from G (B) G

11)

F covers G	G covers F	
✓	✓	F = G
✓	x	F ⊃ G
x	✓	F ⊂ G
x	x	F & G not comparable

12) Properties of decomposition:
- Loss less join decomposition
- Dependency preserving decomposition.

13) Lossless join decomposition :-
Relational schema R with instance r decomposed into sub relations $R_1, \dots R_n$
- In general $R_1 \bowtie R_2 \bowtie \dots \bowtie R_n \supseteq R$ (⋈ : Natural join)
- if $R_1 \bowtie R_2 \dots \bowtie R_n = R$ then lossless join decomposition
- if $R_1 \bowtie R_2 \bowtie \dots \bowtie R_n > R$ then lossy join decomposition. (has extra tuples)

NOTE: ⋈ works on equality of common attr.

14) Conditions for lossless decomposition
- Union of subrelations must contain all attributes.
- common attribute must be present & must be key for atleast 1 relation.

15) Dependency preserving decomposition:
Relational schema R with FD set (F) decomposed into subrelations $R_1, R_2, \dots R_n$ with FD sets $F_1, F_2, \dots F_n$ respectively.
- In general, $\{F_1 \cup F_2 \cup \dots F_n\} \subseteq F$
- if $\{F_1 \cup F_2 \cup \dots F_n\} = F$ then dependency preserving decomposition.
- if $(F_1 \cup F_2 \cup \dots F_n) \subset F$ then not dependency preserving decomposition

16) If some FD of F not member of FD of subrelation then not FD preserving.

17) Minimal cover / canonical cover of FD
- Split FD's so that RHS contains 1 attr.
- One by 1 check if this FD is removed, can it be derived from remaining FDs if yes then remove it.
- Remove Left redundancy in $AB \to C$ if B^+ contains A then remove A if A^+ contains B then remove B.

NOTE:- $AB \to C$ can't be split into $A \to C$ & $B \to C$

18) Minimal set is not unique.

19) Minimal set is equivalence to given FD set.

20) If any FD is removed, don't consider it in further procedure (point 17)

Normal Forms:

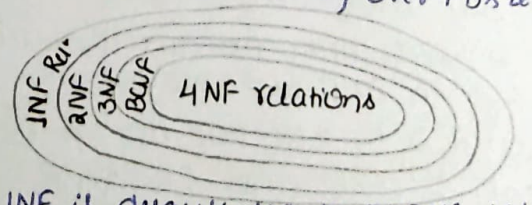
1) used to identify degree of redundancy

2) Redundancy in Relation R may be due to
- non-trivial FD's $X \to Y$ (Single valued dependency) TO eliminate, decompose relation into BCNF.
- Non-trivial MVD's $X \to\to Y$ (Multivalued dependency) TO eliminate, decompose relation into 4NF.

3) Normal Forms :
1NF → 2NF → 3NF → BCNF → 4NF
(1NF → 3NF: Defined over single valued dependencies (FDs) $X \to Y$; BCNF → 4NF: defined over MVDs $X \to\to Y$)

4) BCNF rel" → 0% redundancy over FD's but may exist over MVD's
4NF → 0% redundancy over FD's & MVD's

5)



6) 1NF is default NF of RDBMS tables.
→ Rel" R is in 1NF iff NO Multivalued attribute nor composite attr. in R.
→ If multivalued attr. exist, combine it into CK to convert into single valued
Emp(eid, enom, phno) ← Multivalued attr.
Make {eid phno} CK.

NOTE: Every RDBMS table is in 1NF bcoz by definition all attributes are simple & single valued.

7) $X \to Y$ FD forms redundancy iff
- Non trival FD & - X is not superkey

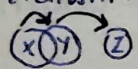
8) $X \to Y$ FD not forms redundancy iff
- Trivial FD OR - X is superkey

9) $X \to Y$ nontrivial FD with X not superkey

	1NF	2NF	3NF	BCNF	4NF
- Proper subset of CK → Non Prime	✓	X	X	X	X
- Non Prime attr. → Non prime attr.	✓	✓	X	X	X
- Proper subset of CK + Non prime attr → Non prime attr	✓	✓	X	X	X
- Proper subset of CK → proper subset of Other CK	✓	✓	✓	X	X

10) Partial dependency: proper subset of CK → Non prime attribute
↳ 2NF failure

11) Transitive dependency: Non prime attr. transitively determined by super key.



X → super key
Y → Not SK
Z → Non prime attr.

↳ Failure of 3NF.

12) 2NF: Rel" R is in 2NF iff NO partial dependency in R exist.

13) 3NF: ∀ non trivial FD $X \to Y$
- X must be super key of R or
- Y must be prime attribute of R

14) BCNF: (Boyce Codd Normal form)
∀ non trivial FD $X \to Y$, X must be SK.

15) Relation R with NO non trivial FD's is always in BCNF but may not in 4NF

16) Relation R with only 2 attributes is always in BCNF & in higher Normal forms 4NF, etc.

17) Rel" R with only simple (not composite) CK's, always in 2NF but may not in 3NF

18) Rel" R with only prime attributes is always in 3NF but may not in BCNF.

19) If Rel" R is in 3NF but not in BCNF then
- at least 2 compound CKs in R
- Overlapped CK's in R (have common attr)
- Proper subset of CK → Proper subset of other CK

20) 3NF is most accurate NF.

21) If $X \to Y$ FD cause redundancy.
Form 1 table with attr X^+ & 1 table having remaining attr with CK of X

NOTE: Prime attributes: attr. that are part of CK's
Remaining → non prime attributes.

→ R(ABCD) $B \to D$ is partial dependency
- Form 1 table with B^+ attributes, B is key of this table.
- 1 table having remaining attr with B. AB is key of this table.

22) Steps to decompose rel" into 3NF:
- Find CKs
- Decompose & create table for all FD's having partial dependency 1 by 1
- Divide FD's into subrelation.
- Again check for partial dependency in subrel", if there decompose else check for transitive dependency.

23) Directly checking for 3NF without checking 2NF.
∀ FD $X \to Y$, X is SK or Y is prime attr. If not decompose.

24) Directly check for BCNF using condition in point 14.

25) While checking partial dependency, we need to consider FD's which can be deduced from given FD's also.
TRICK: If 'AB' is CK find closure of each of its subset i.e A^+ & B^+ and check if it derives non prime attribute or not.

26) Remove FD's which contains attri. which are not part of relation.

27) If there is no FD on a relation, it is in BCNF.

DB design goal	1NF	2NF	3NF (Most Accurate)	BCNF	4NF
1) Lossless join decomposition	Possible for every rel" decomposition	Possible for every rel" decomposition	Possible for every rel" decomposition	Possible for every rel" decomposition	May not possible for every rel" decomposition
2) Dependency preserving	Possible for every rel" decomposition	Possible for every rel" decomposition	Possible for every rel" decomposition	May not possible for every rel" decomposition	May not possible for every rel" decomposition
3) 0% Redundancy	NO	NO	NO	Yes over FDs NO over MVDs	Yes over FDs and MVDs

Transaction & Concurrency Control:

1) Transaction is a set of logically related operations to perform unit of work.
OS → Process , DBMS → Transaction

2) Database → Files (Tables) → Blocks (Pages) → Records

3) Data item:- shared resource by many transaction. (may be DB/File/Block/Record)

4) Concurrency control is method to avoid errors becoz of concurrent execution of 2 or more trans over same DB.

5) Degree of concurrency: No. of users that can use DB simultaneously.

6) In Flat file system (OS files)
- each file is a resource
- less degree of concurrency
- easy to maintain locks.

7) In DBMS concurrency control:
- each record is a resource.
- More degree of concurrency.
- complex to maintain locks.

8) Thus, DBMS uses Multilevel (DB/File/Block/Record) concurrency control depending on trans requirement.

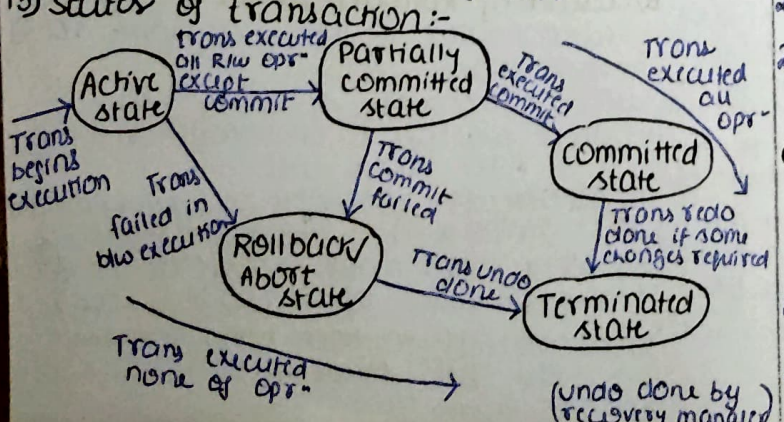
9) Degree of concurrency
at record level > over Block level > At file level > at DB level

10) Complexity of Lock management follows same order.

11) Main operations in trans: Read(x), Write(x)
Both are atomic.

12) TO preserve integrity (correctness), trans must satisfy ACID property.
A: Atomicity } Maintained by Recovery
D: Durability } management system of DBMS
I: Isolation } Maintained by concurrency control component of DBMS
C: Consistency } Maintained by user

13) States of transaction:-



14) Atomicity: either execute all opr- of trans including commit or execute none opr- of trans.

15) Durability: After trans completes successfully, the changes it has made to DB persist even if there is system failure.
& Trans. should able to recover under any case of failure.

16) Failure may be power failure, s/w crash, disk crash or concurrency control protocol may kill trans.

17) For disk crash recovery, RAID (Redundant array of independent disk) is used.

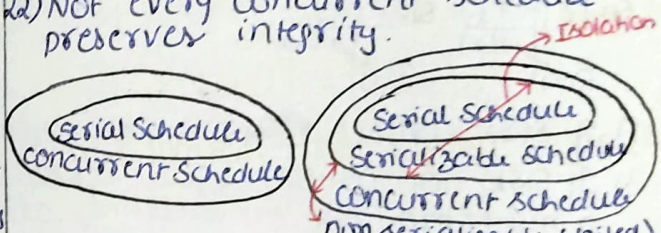
18) Consistency: DB must be consistent before & after the trans.

19) DBMS can't detect inconsistencies due to errors in the program logic. It's user responsibility.

20) Isolation: Concurrent execution of 2 or more trans result must be equal to result of some serial execution of trans. (Serializable schedule)

21) Every serial schedule is always consistent.

22) Not every concurrent schedule preserves integrity.



23) If T1 has m opr" & T2 has n opr-
Total no. of schedules possible = (m+n)!/m!n!
- For K no. of trans, K! serial schedules possible

24) Classification of schedules based on serializability:-
conflict ser. , view ser. schedule.

25) condition for conflict: Atleast 2 trans with common data, & atleast 1 write operation.

26) Conflict can be RW, WR or WW.

27) Conflict equal schedule:-
S1 ← swap adjacent non conflict pair → S2

Other way-
Each trans Ti in S1 must be same in S2 and every conflict pair preceding in S1 must be same as in S2.

28) S1 & S2 must be with same set of trans.

29) If S1 & S2 conflict equal then precedence graph must be same.

30) If $S_1 S_2$ schedule don't have same precedence graph ⇒ $S_1 S_2$ not conflict equal

31) If same precedence graph having cycle ⇒ schedules may/may not conflict equal.

32) If same precedence acyclic graph ⇒ S_1 S_2 conflict equal.

33) Conflict Serializable Schedule: some serial schedule must be conflict equal to given schedule.

NOTE: Precedence graph of serial schedule is acyclic.

34) Schedule S is conflict serializable iff precedence graph of S is acyclic.
- conflict equal serial schedule is topological orders of precedence graph of S.

35) NO of topological orders of S precedence graph = NO. of serial schedule which are conflict equal to schedule.

36) View equal Schedule: $S_1 S_2$ schedules are view equal iff
- Initial read - if Ti reads x initially in S_1 then in S_2 also Ti must initially read x
- Final write
- updated read

S_1: Ti | Tj : W(x) : R(x)

S_2: Ti | Tj : W(x) : R(x)

37) If $S_1 S_2$ conflict equal ⇒ View equal

38) If $S_1 S_2$ not conflict equal schedule ⇒ may/may not be view equal.

39) View serializable schedule: some serial schedule is view equal to given schedule

40) Finding view equal serial schedules-
- Find trans which has done final write.
- Find trans done initial read.

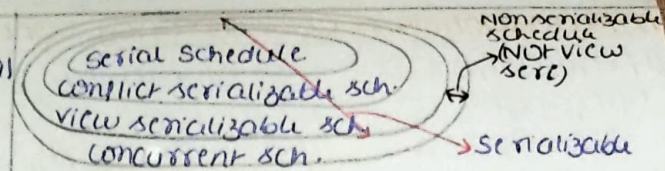
Given Sch.	View equal Sch
X: Ti (Tj)FW	Ti → Tj
Y: Ti Tj (Tk)FW	(Ti, Tj) → Tk
Z: (Ti)FW	Every serial is view equal

Given schedule			view equal
data item	Initial Read	write	
X	Ti	Tj	Ti → Tj
Y	Ti Tj	Tj	Ti → Tj
Z	Ti Tj	Tj Tk	Ti → (Tj Tk) & Tj → Tk
A	Ti Tj	NO write	All serial sch is view equal

- Find updated read
  - Wi(x) → Rj(x) & NO other trans writes x ⇒ Ti → Tj, other trans may come in blw
  - Wi(x) → Rj(x) & let Tk also writes x ⇒ Ti → Tj, Tk must not come in blw.

41) If conflict serializable ⇒ view serializable

42) if not " " ⇒ may/may not view ser.



43) Conflict serializable sch testing condition is 'sufficient but not necessary' condition to test serializability.

Precedence graph of S is acyclic ⟶ S is serializable

TC = O(n²)

44) View ser. sch. testing condition is 'both sufficient & necessary'

view serializable ⟷ S is serializable

TC = O(2ⁿ)

---

1) Concurrent execution may leads to the problem of irrecoverable, cascading rollback, lost update prblm even though schedule is serializable.

2) Irrecoverable schedule Prblm :-

Ti	Tj
W(x)	
	R(x)
commit	

iff Tj reads data x updated by Ti & commit of Tj before Ti.

fail
undo

3) Dirty Read (Uncommitted Read): Tj reads data x updated by uncommited trans Ti. (as above)

4) Recoverable sch (Protocol conditions to solve irrecoverable prblm) :
Sch S is recoverable iff:
- NO uncommitted read in schedule.
- if Tj reads x updated by Ti then commit of Tj must delay until commit/rollback of Ti.

5) Recoverable sch conditions does not avoids cascading rollback prblm, lost update prblm.

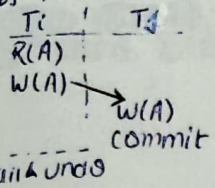
6) Cascading Rollback prblm: failure of 1 trans forcd to rollback some set of other trans.
- occurs because of dirty read.
- Disadv: wastage of CPU execution time & IO Access cost.

7) Cascadeless Rollback Schedule (Protocol) if Ti writes x then read of updated x by some other trans should delay until commit/rollback of Ti. (NO Dirty read)

8) Cascadeless rollback condition not avoids lost update prblm.

- NO Dirty Read ⇒ Recoverable, NO Cascade Rollback
  (NO over MVDs | Und MVDs)

9) Lost Update Prblm: if trans Tj writes X which is already written by uncommitted trans Ti, lost update prblm can occur.



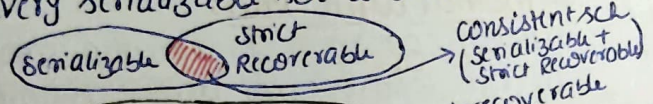
updation done by Tj is lost.

Sol": Delay write of Tj until commit / Rollback of Ti

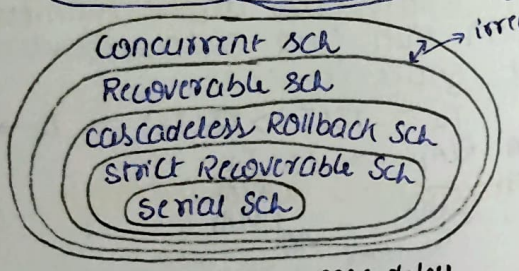
10) Strict Recoverable schedule: if Ti writes X then Read(X)/write(X) by Tj must delay until commit rollback of Ti.

- It prevents irrecoverable, cascade Rollback & lost update prblm.

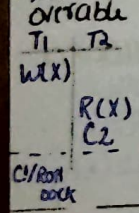
11) NOT every strict recoverable sch is serializable NOT every serializable sch is strict recoverable



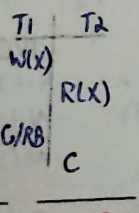
12)



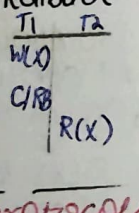
13) Irrecoverable



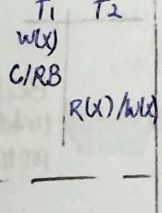
Recoverable



Cascadeless Rollback



Strict Recoverable



Concurrency control protocols :-

1) Goal: should not allow to execute any schedule which is
   - Non serializable schedule
   - Non strict recoverable schedule.

2) Protocols:
   - Locking protocols
   - Timestamp ordering protocol

3) Schedules accepted to be executed by protocol must be serializable & strict Recoverable.

Locking Protocol:

1) Lock:- is a variable associated with a data item that describes the status of the data with respect to possible operations that can be applied to it.

2) Binary lock: Lock(X), Unlock(X) (2 states)
   - Less degree of concurrency
   - Can't perform concurrent read also.

3) We use Shared / Exclusive lock (More degree of concurrency)

4) Shared lock: Read only lock (S)

5) Exclusive lock: Read-write lock (X)

6) When there is no lock (shared/exclusive) on a data item X then only exclusive lock can be acquired by a transaction

Ti	T2
S(X)	
	X(X) ← Denied

7)

data item	Ti holding S	Ti holding X
Tj requesting S	Yes	NO
Tj requesting X	NO	NO

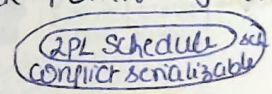
8) Binary locking + 2PL } Guaranteed serializability
   Shared exclusive lock + 2PL }

9) Two phase locking protocol (2PL): Trans(T) allowed to request lock on any data item in any mode until first unlock done by T.
   → Locking phase (Growing ph)
   - Lock request not allowed in unlocking phase (shrinking phase)

Lock(A) Lock(B) Unlock(B) Unlock(A)
→ Lock point of trans (last lock & 1st unlock)

10) If schedule not conflict serializable schedule then schedule not allowed by 2PL.

11) If schedule(S) allowed by 2PL then S is conflict serializable schedule and
conflict equal serial schedule = Topological order of precedence graph = order of lock points of transaction.

12) 

13) Not every conflict seri schedule is 2PL

14) Lock upgrading 2PL: if trans Ti required R(A) followed by W(A) then trans first request for shared lock(S) & just before W(A) upgrade to exclusive lock(X) && lock upgradation allowed in locking phase of trans

	Ti	Tj
Growing phase	S(A)	
	R(A)	} other trans can use A in shared lock mode
	X(A)	
	W(A)	

Adv: More degree of concurrency

15) Limitations of 2PL:
- Not free from irrecoverable, cascading rollback, lost update prblm, deadlock, starvation.

16) 2PL ≈ Serializability

17) Strict 2PL – correct protocol for concurrency control

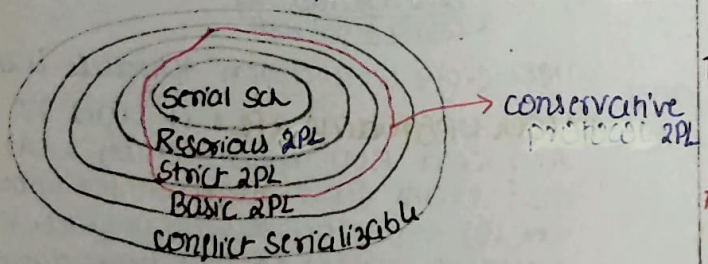
C

18) Strict 2PL:
Basic 2PL → Serializable
+
Strict Recoverable: All exclusive locks of tran must hold until commit/Rollback of trans.
Ex: S(A) X(B) S(C) | U(A) U(C) commit U(B)
growing phase | shrinking phase

19) Rigorious 2PL: = Basic 2PL + (→ strict Recoverable)
All locks (shared/exclusive) of trans must hold until commit/rollback of trans.
Ex: S(A) X(B) S(C) commit U(A) U(B) U(C)
↑ lock point

20) Conservative 2PL: = 2PL +
Deadlock prevention: trans must lock all required data item in initial stage before trans begins execution.
- Its not free from starvation, & not efficient utilization of data item.

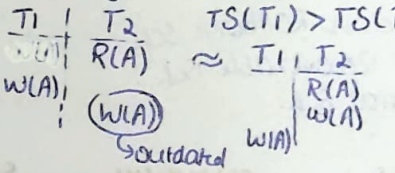
	Basic 2PL	strict 2PL	Rigorious 2PL	conservative 2PL
① Guaranteed Serializability	yes	yes	yes	yes
② Guaranteed strict Recoverable	NO	yes	yes	NO
③ Free from deadlock	NO	NO	NO	yes
④ " " starvation	NO	NO	NO	NO
⑤ Equal serial schedule	based on lock point order	based on lock point order	based on commit order	based on lock point order



Tree based protocol: (Graph based)
1) use only exclusive lock
2) NO 2PL, can unlock anytime
3) completely avoids deadlock
4) ensures conflict serializability.
5) don't ensure recoverability and cascadelessness.
6) Any data can be locked at most once by each transaction.
7) 1st lock is on any data item, after words, a data item can be locked only if parent is locked

Timestamp based protocol:
1) Trans get timestamp on arrival.
2) If conflict is in order of timestamp then it's executed.
3) conflict equivalance to trans in their timestamp order.
4) NO deadlock.
5) conflict serializable.
6) don't ensure recoverability and cascadelessness.
7) if conflict is not in timestamp order trans rollsback & start with new timestamp.

Thomas write Rule:
1) Modification of timestamp protocol.
2) First find serial order (in timestamp order) then find outdated write and ignore it.



If TS(T1) > TS(T2) ⇒ T2 → T1

	Allowed	Not allowed
Basic timestamp protocol	All opr" where T2 occurs before T1 & R1(X) R2(X)	R1(X) W2(X) W1(X) R2(X) W1(X) W2(X)
Thomas write Rule	• All opr" where T2 occurs before T1 • outdated writes W1(X) W2(X) • R1(X) R2(X)	R1(X) W2(X) W1(X) R2(X)

Procedure for checking serializability:
- check for conflict ser.
  if conflict ser ⇒ then view ser
  if not conflict-ser
  ↳ check for blind write (writing without reading)
  if no blind write ⇒ no view ser.
  if blind write ⇒ check for view ser.

NOTE: when system crashes, check is performed from crash point to previous checkpoints. (after checkpoint)
Trans which are normally committed are added to redo list.
Trans which are terminated without commit, are added to undo list.
- when a checkpoint is created, trans which are committed are added to stable storage and removed from log.
- If checkpoint is not used then all the trans committed are redone & aborted trans are undone.

... NO cascade Rollback

Relational Algebra:

Query lang
- Procedural: Formulation of what data is retrieved & how data is retrieved from DB. Relational Algebra
- Non Procedural: Formulation of what data is retrieved from DB table. Relational calculus (uses 1st order logic predicate calculus)
  - Tuple Relational Calculus (TRC)
  - Domain Relational Calculus

1) SQL Queries close to TRC formulas. QBE (Query by example) close to DRC formulas

2) Basic operators in Relational algebra

Unary operators:
- π: Projection
- σ: Selection
- ρ: Rename

Binary operators:
- $\times$: Cross product
- $\cup$: Union
- $-$: Set difference (minus)

Derived operators:
- $\cap$: Intersection (using $-$)
- $\bowtie$: Join (using $\times$ σ π)
- $\div$ or $/$: Division (using π $\times$ $-$)

NOTE: Arity – No. of columns in table

3) Cross Product ($\times$): $R \times S$ results all attributes of R followed by all attr of S and each record of R pairs with every record of S.

R(A,B,C)	S(C,D)	R×S(A,B,C,C,D)
Arity: x	Arity: y	Arity: (x+y)
n distinct record	m distinct record	Tuple cardinality = n*m

→ If either R or S is empty then R×S results an empty record set.

4) Joins:

@ Natural Join ($\bowtie$):

$R \bowtie S = \pi_{\text{distinct Attributes}}(\sigma_{\text{equality of common attributes}}(R \times S))$

Ex: $R \bowtie S = \pi_{ABCD}(\sigma_{R.C=S.C}(R \times S))$

$T_1(A\ B\ C)$, $T_2(B\ C\ D\ E)$

$T_1 \bowtie T_2 = \pi_{ABCDE}(\sigma_{T_1.B=T_2.B \wedge T_1.C=T_2.C}(T_1 \times T_2))$

→ Natural join equal to cross product if no common attribute.

(b) Conditional Join ($\bowtie_c$)

$R \bowtie_c S = \sigma_c(R \times S)$

Ex: $R \underset{R.C>S.C}{\bowtie} S = \sigma_{R.C>S.C}(R \times S)$

→ Arity of $R \bowtie S \leq$ Arity of $R \times S$

Arity of $R \bowtie_c S$ = Arity of $R \times S$

(c) Outer Joins:-

i) Left Outer join ($\overset{L}{\bowtie}$)

$R \overset{L}{\bowtie} S = (R \bowtie S) \cup$ (Records of R those failed join condition with NULL value for remaining attributes)

ii) Right Outer join ($\overset{R}{\bowtie}$)

$R \overset{R}{\bowtie} S = (R \bowtie S) \cup$ (Records of S those failed join condition with NULL value for remaining attributes)

iii) Full Outer join ($\overset{F}{\bowtie}$)

$R \overset{F}{\bowtie} S \equiv (R \overset{L}{\bowtie} S) \cup (R \overset{R}{\bowtie} S)$

→ Outer join can be applied to conditional join also.

$R \underset{R.C>S.C}{\overset{L}{\bowtie}} S$

→ Join Queries:

1) $\sigma_c(R \times S)$ / $R \bowtie_c S$ or $R \bowtie S$ } Some / Any / Atleast One

Retrieve A of R those are more than some B value of S.

$\pi_{R.A}(\sigma_{R.A>S.B}(R \times S))$ or $\pi_{R.A}(R \underset{R.A>S.B}{\bowtie} S)$

2) Every / All

$A > \text{every } B = A - (A \leq \text{some } B)$ (some: $\bowtie_c$)

Retrieve A of R those are more than every B of S.

$\pi_A(R) - \pi_{A.A}(R \underset{R.A \leq S.B}{\bowtie} S)$

3) Max / Min

Max salary = sal ≥ every sal

Min salary = sal ≤ every sal

4) Max / Min value for each group of record

Retrieve eid who gets max salary for each dept.

$\pi_{eid}(Emp) - \pi_{eid}(Emp \underset{sal<S \wedge dno=D}{\bowtie} \rho_{I,S,D}(Emp))$

→ Rename (ρ):

- $\rho(Temp, stud)$ stud table is renamed as Temp
- $\rho_{A,B,C}(stud)$ Attributes of stud are renamed as A, B, C

- Renaming some attributes of table
  $\rho_{(stud)}$ id→i, sal→s

⇒ Set operators:

∪: Union
−: set difference } Basic operators
∩: intersection → Derived operator

1) To apply set operators, relations must be union compatible i.e.
- No. of attributes in R = No. of attr. in S
- Domain of each attribute of R must be same domain for S attributes respectively

$\pi_{A,B}(I) \overset{\cup}{\cap} \pi_{A}(I) \propto \mid \pi_{Sid,nome}(\,) \overset{\cup}{\cap} \pi_{Sid,DOB} \propto \mid \pi_{Sid,nome}(\,) \overset{\cup}{\cap} \pi_{nome,Sid}(\,) \propto$

2) R∪S, R∩S, R−S
- Resulted relation name & schema is same as R
- Distinct tuples.

3) ∪: either from R or from S (OR)
∩: records from R and S (AND)
−: Records from R but not from S (But Only / Only)

4) Various way to perform intersection
R(A,B) S(C,D)
- $R \cap S = R - (R - S) = S - (S - R)$
- $R \cap S = \pi_{A,B}(\sigma_{R.A=S.C \wedge R.B=S.D}(R \times S)) = \pi_{A,B}(R \bowtie_{R.A=S.C \wedge R.B=S.D} S)$
- $R \cap S = R \bowtie_{A,B} \rho_{(S)}$

5) Only (−) ∝ = All − some not ∝
6) A and B = some A ∩ some B

⇒ Queries to express atleast two, only two, atmost two.

Enroll(Sid, Cid) stud(Sid Snome)

1) Retrieve Sid enrolled in atleast two courses
$\pi_{Sid}(Enroll \bowtie_{Sid=S \wedge Cid \neq C} \rho_{S,C}(Enroll))$ ↳ take 2 instance of relation

2) Retrieve Sid enrolled in atleast 3 courses ↳ take 3 instance
$\pi_{T_1.Sid}(\sigma_{T_1.Sid=T_2.Sid \wedge T_2.Sid=T_3.Sid \wedge T_1.Cid \neq T_2.Cid \wedge T_1.Cid \neq T_3.Cid \wedge T_2.Cid \neq T_3.Cid}(\rho_{(T_1,Enroll)} \times \rho_{(T_2,Enroll)} \times \rho_{(T_3,Enroll)}))$

3) enrolled only one course
= enrolled atleast 1 course − enrolled atleast 2 courses

4) Only 2 course = atleast 2 − atleast 3 course

5) atmost 1 course = all student − atleast 2 course

6) atmost 2 course = all − atleast 3 course

⇒ Division Queries : (/ or ÷) (pair with every)

R(X,Y) S(Y)

$\pi_{XY}(R) / \pi_{Y}(S)$: Result set contains value of X such that for every value of y in S there is a tuple (x,y) in R.

① Retrieve Sid enrolled in every course
$\pi_{Sid,Cid}(Enroll) / \pi_{Cid}(course)$

② Expansion of division:-
Sid enrolled for every course = Sid enrolled in some course − Sid not enrolled in every course
$= \pi_{Sid}(Enroll) - \pi_{Sid}(\pi_{Sid}(Enroll) \times \pi_{Cid}(course) - \pi_{Sid,Cid}(Enroll))$

③ $\pi_{AB}(R)/\pi_{B}(S) = \pi_{A}(R) - \pi_{A}((\pi_{A}(R) \times \pi_{B}(S)) - \pi_{AB}(R))$
$\pi_{ABCD}(R) / \pi_{CD}(S) = \pi_{AB}(R) - \pi_{AB}((\pi_{AB}(R) \times \pi_{CD}(S)) - \pi_{ABCD}(R))$

④ Rel" R with n distinct tuples & X set of attributes, Rel" S with m distinct tuples (m>0) & Y set of attributes.
- To apply R/S, required condition is X ⊃ Y
- Attributes in result set = X − Y
- cardinality of R/S → [0 to ⌊n/m⌋ tuples if m>0] (↳ atmost n tuples if m=0)

n → cardinality of R, m → cardinality of S

⇒ Min-Max records in result of natural join

1) R(A B C), S(A C)
n tuple, ↳ m tuple

R		S	
A	B	A	C
1	A	1	a
2	B	1	b
3	B	2	c

1:M

In R⋈S,
Max = m tuple
Min = { m tuple if A of S is FK
      { 0 if no FK

2) common attr key for 2nd table
R(A B C) n^, S(B D E) m

On R⋈S ← Many side
max = n tuples
min = { n if FK
      { 0 if no FK

R			S	
A	B	C	B	D
a_1	2		2	
a_2	2		3	
a_3	3		4	

M:1

3) common attr key for both table
R(A B C) S(A D E)
Max = min(n,m)
Min = { min(n,m) if FK
      { 0 if no FK

R		S	
A	B	A	D
a_1		a_1	
a_2		a_2	
a_3		a_3	

1:1

4) common attr not key for both table
R(A B) S(D B)
Max = n * m
Min = 0

A	B
a_1	1
a_2	1

D	B
d_1	1
d_2	1

Rel^n R with n distinct tuples & S with m distinct tuples.

Operation	cardinality (min - max)	ab=ba commutative
$\pi_A(R)$	n	NO
$\sigma_C(R)$	0 to n	Yes
$R \times S$	n*m	Yes
$R \bowtie_C S$	0 to n*m	Yes
$R \bowtie S$	n to n*m	NO
$R \bowtie S$	m to n*m	NO
$R \bowtie S$	max(n,m) to m*n	Yes
$R \cup S$	max(n,m) to n+m	Yes
$R \cap S$	0 to min(n,m)	Yes
$R - S$	0 to n	NO
R / S	0 to $\lfloor n/m \rfloor$ if m>0; 0 to n if m=0	NO

- $\pi_A(\pi_{AB}(R)) \neq \pi_{AB}(\pi_A(R))$ NOT commutative

SQL (Structured Query Language)

1) Sub languages of SQL: DDL, DML, DCL, TCL

2) Data Definition Lang (DDL): used to define/modify structure of DB table & integrity constraints (keys).
- CREATE - DROP - ALTER
- TO define PK, UNIQUE, FK, NOT NULL, etc

3) Data Manipulation Lang (DML): used to modify data records & to access data.
- INSERT - DELETE - UPDATE - SELECT

4) Data control lang (DCL): used to grant & revoke data access to user for security.
- GRANT - REVOKE

5) Transaction control lang (TCL): used for concurrency control.
- COMMIT - ROLL BACK - CHECK POINT

6) SQL vs RA

Select distinct A_1, A_2 ⇒ Projection (π)
From R_1, R_2 ⇒ Cross Product ($\times$)
where p; ⇒ selection (σ)
$\equiv \pi_{A_1,A_2}(\sigma_p(R_1 \times R_2))$

NOTE: In RA everything is considered as sets can't have duplicate value in result. In SQL, duplicate value may occur.

NOTE: In SQL, an attribute 'Rowid' is defined by default as a default PK.

7) Basic SQL clauses:
⑤ Select [⑥ Distinct] $A_1, A_2 \cdots A_n$
① From $R_1, R_2, \cdots R_m$
② [where condition]
③ [group by attributes
④ [Having condition]]
⑦ [Order by attributes [DESC]];
[] denote optional.

8) Aggregate fu" supported by SQL: (5)
COUNT(), SUM(), AVG(), MAX(), MIN()

9) Aggregate fu" computes aggregation of NON-NULL values. NULL values are discarded.

10) COUNT(*) : no. of records
COUNT(A) : no of non NULL values of attrA
COUNT(DISTINCT A): no of distinct non null values of A

11) GROUP BY clause:
used to grp records based on attr value.
NOTE: NULL is also made 1 group
if group by clause used then:
- every attribute of group by clause must be in select clause.
- Allowed aggregation fu" on [illegible]
- Not allowed to select any other attribute in select clause.

NOTE: aggregate fu" is computed for each grp

12) HAVING clause:
- comes only with group by clause
- WHERE checks condition for each record. HAVING checks condition for each group.
- condition must be over aggregate fu" or over functions like some(), every(), etc
- Direct attribute comparision not allowed.

Nested Queries:

① Nested Query without correlation:
- inner query independent of outer query
- " " computes first & only once
- execution flow bottom to top.
- allowed in FROM, WHERE, HAVING clause of outer query.

② correlated Nested Query:
- Inner query uses attributes from the outer query table.
- Select *
① From R
③ where (Inner ② Query) < 3;
① group by dno
③ having Avg(sal) > ② (inner query)

for each record of relation R, inner query is computed then outer query where is tested.

- correlation allowed in WHERE & HAVING clause of outer query.
- If correlation in WHERE clause, no. of times inner query computed = no. of records in outer query FROM clause.
- If correlation in HAVING clause, no. of times inner query computed = no. of group of outer query

NOTE: In SQL, 'EXCEPT' is used as (—) minus

(1) Retrieve eid who gets 2nd highest sal.

ρ(Temp, $\pi_{eid,sal}$(Emp $\bowtie_{sal<s\ I.s}$ ρ(Emp))

π_{eid}(Temp) − π_{eid}(Temp $\bowtie_{sal<s\ I.s}$ ρ(Temp))

to avoid recomputation of Temp, we use WITH in SQL.

- WITH clause: Sub query result of WITH clause can be reused many times.

```
* Select T1.Eid
  From Emp T1
  Where (Select count(distinct T2.Sal)
         From Emp T2
         Where T1.Sal < T2.Sal) = 1 ; 2nd Max
                                =0  Max
                                =2  3rd max
                                ≤1  1st & 2nd Max
```
(> will give min)

⇒ Functions used for Nested Query:-

If inner query results more than 1 record to compare with outer query instead of using (>, =, <), condition required to use fu"

- IN / NOT IN
- ANY
- ALL

} Preferred for NON-correlated queries

- EXISTS / NOT EXISTS } preferred for correlated query

(1) IN:

- used for membership test.
- x in Y is True if x value member of Y.
- x in {Empty set} : False
  x not in {Empty set} : True
- Equi Join ($\bowtie_=$) queries (Natural join or conditional join with equal condition) can expressed using IN function.

(2) ANY / ALL fu":

- preceded by comparison operator
  <, <=, >, >=, =, <> not equal to
- x opr ANY {Empty set} False
  x opr ALL {Empty set} True

- 'ANY' fu" used to represent conditional join.
- '=Any' equal to 'IN' fu"
  '<>ALL' equal to 'NOT IN' fu"

(3) EXISTS fu":

- used to test result of inner query empty or not empty.
- EXISTS (inner query) returns True if result of inner query not empty i.e contains atleast 1 record.
- EXISTS fu" can be used for conditional join queries.
  $\bowtie_c$ = EXISTS fu" in correlation.
  EXISTS = some / Any / atleast

$\pi_{R.A}$(R $\bowtie_{R.A>S.B}$ S) ≡

```
Select A
from R
where exists ( select *
               from S
               where R.A>S.B);
```

NOTE: All columns must be of some length

```
Select AVG(A), B
From R;
```
✗ Invalid

* Avoid using aggregate fu" in WHERE clause, it does not give error but computes for each record.

→ Order by clause:

- By default it sorts in ascending order.
- NULL values is always at last.

NOTE: (A,B) = ANY (set of values) is wrong
(A,B) IN (set of values) ✓

In 2 field comparison, IN can't be replaced by =ANY.

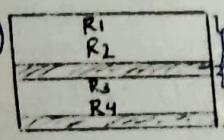
NOTE: >All can be replaced with >MAX
>ANY " " " >MIN

# Limitations of Flat File System (OS file)	Advantage of DBMS
• Too complex to develop & manage application programs.	• Its easy because of data independency (user access data using SQL interface)
• More I/O (access) cost	• DB indexing reduces I/O cost.
• Less degree of concurrency.	• More degree of concurrency.
• complex to manage non-redundant data	• using normalization of DB, its easy.
• Difficult to implement different levels of access control (security)	• using views, easy to implement different levels of access control.

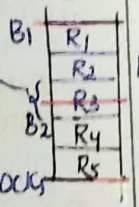
File Organization and Indexing

1) DB is collection of Files (Tables).
File is collection of Blocks (Pages)
Block is collection of records.

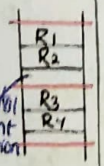
2) Records of the DB file can be:
(i) Fixed length records - each attribute is of fixed length.
(ii) Variable length records - some attr size is not fixed.
Ex: D char(100) ← Fixed size 100B
E text ← Variable length

3) 

B1 → Block header ← if not given take it 0
B2 (offset table to access record)

4) Records organization into DB file can be:
(a) Spanned organization: Record allowed to span in more than 1 block.


- Adv: Less internal fragmentation.
- Dis: More access cost as multiple blocks need to be accessed.

(b) Unspanned organization:
Complete record must be in 1 block.


- Adv: Less access time
- Dis: More internal fragmentation

5) Unspanned organization preferred to organize DB file with fixed length record
- Spanned organization preferred to organize DB file with variable length record

6) Block factor of DB file: Max possible records which can be stored in 1 block.
Block size = B bytes
Block Header = H bytes
Record size = R bytes (fixed length)
Block factor = $\left\lfloor \frac{B-H}{R} \right\rfloor$ records/block
(because unspanned org used) | For spanned org, BF = $\left\lceil \frac{B-H}{R} \right\rceil$

7) Indexing: reduces access cost (I/O cost)

8) Access cost: no. of disk blocks required to transfer from disk to Main memory in order to access required data.

9) Ordered file organization: All records in a file are ordered on some attribute called ordered field, (its set by DBMS (need not be key))
Remaining attr. are called unordered field
- Binary search can be performed on ordered field.
- A field on which majority operations are performed is chosen as ordered field.

10) Unordered file organization:
- No ordering done.
- Linear search performed.
- Insertion easier.

11) I/O cost to access record without indexing:
n → no of blocks in DB file.
- using ordered field: $\lceil \log_2 n \rceil$ blocks (WC)
- using unordered field: n blocks (WC)

12) In index file, search key & block pointer is stored.

13) I/O cost to access record with index:
- using 1st level index = $\lceil \log_2 m \rceil + 1$ block
m → no of block in index file
- using Multilevel indexing = (k+1) blocks (k-level index)

14) Multilevel index: index to index until last level has 1 block of index

15) Each entry in index file has 2 fields
<search key, pointer>

16) Block factor of index file = max possible index entries in each block
$= \left\lfloor \frac{B-H}{K+P} \right\rfloor$ entries/block

17) Block factor for DB file = max possible records in each block
$= \left\lfloor \frac{B-H}{R} \right\rfloor$ records/block

B → block size H → block header size
K → search key size P → pointer size
R → Record size

18) Indexing → Dense indexing
→ Sparse "

19) Dense index: For each search key value of DB file there must be entries in index file.
- Its preferred in case of unsorted field.
- No. of index file entries = No. of distinct values of search key in DB file

20) Sparse index: For set of records there must be entry in index file.
- Preferred in case of sorted field.
- No. of index file entries = No. of blocks of DB file

21) Search key :- Field used for indexing
- It may be ordered, unordered, key or non-key.

NOTE: Index is sorted file on search key.
- Index file is always unspanned.

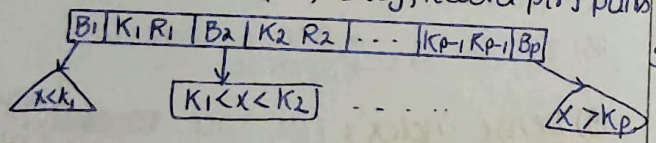
23) Indexing

- Search key: Ordered field
  - Search key: Primary / Alternate key → Primary Index
    - can be dense or sparse (sparse PI is preferred)
    - I/O cost to access record using PI with multilevel index = (k+1) blocks
    - At most 1 PI for any DB file
  - Search key: Non key → clustering Index
    - CI mostly sparse index (if each cluster 1 record then dense also possible)
    - = k + one or more records (there may be repeated value)
    - At most 1 CI
- Search key: Unordered field
  - Search key: Primary / Alternate key → Secondary Index (over key)
    - Always dense index
    - = (k+1) block
    - Many SI possible.
  - Search key: Non key → Secondary Index (over non key)
    - Both dense & sparse possible
    - = k + one or more blocks
    - Many SI possible.

24) Multilevel indexing → B tree, B+ tree } These are index files

#B Tree:

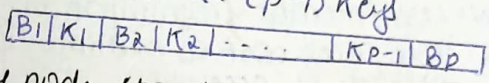
1) Record ptr / data ptr : pointers pointed to DB blocks.
2) Block ptr / Tree ptr / Child ptr: pointers pointed to index node (tree node)
3) Order p : Max possible block pointers which can be stored in a tree node
4) Node structure: p block ptrs, (p-1) [key, record ptr] pairs



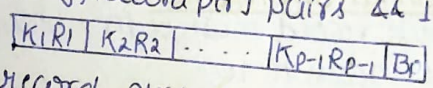
5) Every internal nodes except root must have atleast ⌈p/2⌉ block pointers with ⌈p/2⌉-1 keys and atmost p block ptrs with (p-1) keys.
6) Root can have atleast 2 block ptr with 1 key & atmost p block ptr with (p-1) keys.
7) All leaf nodes must be at same level
8) If a node overflows with insertion, split from middle.
9) All nodes except root must be 50% occupied.
10) If field used for indexing is ordered, we need pointer only to 1st record of block ie sparse index.
- If field used for indexing is unordered, we need pointer to each record ie dense indexing.
11) Adv of B tree: B tree index best suitable for access of any 1 record.
- select * from R where A = 38;
- Best case IO cost : 2 blocks
  Worst case IO cost : (k+1) blocks
  k → Level of tree.
12) Disadv: not suitable for sequential access of range of records.
- select * from R where A ≥ 25 and A ≤ 38
- Estimated IO cost to sequentially access x records = x(k+1) blocks = O(xk)

#B+ Tree:

1) Order p: max possible pointers which can be stored in B+ tree node.
2) Internal node structure:
   p block pointer & (p-1) keys

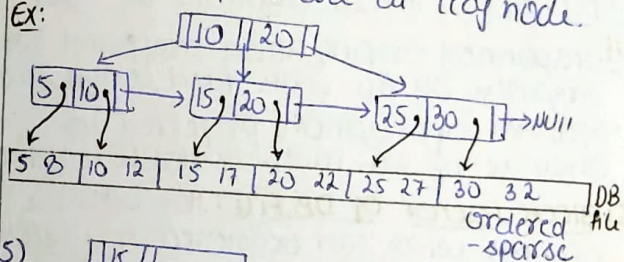


3) Leaf node structure:
   (p-1) [key, record ptr] pairs & 1 block ptr



4) All record ptrs are at leaf node.

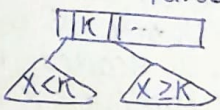
Ex:



Ordered - sparse

5)

Left biasing



Right biasing

6) Every internal nodes except root must have atleast ⌈p/2⌉ block pointers with ⌈p/2⌉-1 keys & atmost p block pointer with (p-1) keys.
7) Root can be atleast 2 block pointers with 1 key & atmost p block pointers with (p-1) keys.
8) Leaf node must contain atleast ⌈p/2⌉-1 keys & atmost (p-1) keys.
9) All leaf nodes must be at same level.

NOTE: Balancing condition (6,7,8,9 pnt) same as balancing condition of B tree (5,6,7)

10) Adv:

- best suitable for random access of any 1 record.

IO cost = (K+1) blocks

K → no. of levels in tree.

- best suitable for sequential access of range of records.

Estimated IO cost to access x records = $(K-1) + \frac{x}{C} + x \approx O(x+K)$

(K−1): no of blocks above leaf node; x/C: no. of blocks of leaf node; x: no. of DB file blocks

C → no. of record ptrs per leaf node

NOTE: In B tree, block pointers of leaf node are wasted (pointing to NULL)

If block size allocated to B tree & B+ tree node is equal then

(i) Order of B tree < Order of B+ tree node

(ii) Total no. of nodes in B tree for n distinct keys > Total no. of nodes in B+ tree for n distinct key (for large n)

(iii) No. of levels in B tree index for n distinct keys > No. of levels in B+ tree

(iv) IO cost in B tree > IO cost in B+ tree

B+ tree index has less IO cost for random access of any 1 record bcoz of less levels.

- Less IO cost for sequential access of range of records

Thus, B+ tree index is preferred then B tree index for DB file indexing.

If order of B tree = order of B+ tree

(i) No. of B Tree nodes for n distinct keys < No. of B+ tree nodes for n distinct keys (for large n)

(ii) No. of levels of B tree index for n distinct keys < No. of levels of B+ tree index for n distinct keys.

(iii) IO cost of B tree < IO cost of B+ tree

NOTE: B tree has more depth, B+ tree has more breadth.

- If in ques, Record ptr & block ptr size not given separately, consider both equal.
- B & B+ trees are balanced.
- while inserting, level can increase by max 1
- deletion, - - decrease by max 1

* If order is p, max no of keys in B/B+ trees of L levels = $p^L - 1$

Relational calculus:

Tuple Relational calculus ← → Domain Relational calculus

1) SQL is based on Tuple relational calculus.

2) A language is said to be relationally complete if it has the capability to express all queries of relational Algebra

3) Unsafe query: A query whose output is possibly infinite. Ex: {T / ~(T ∈ Student)}

4) TRC & DRC has unsafe operations whereas RA don't have.

TRC:

1) Selects tuples from a relation.

2) Syntax: {t / cond(t)}
(t: tuple var; cond: condition)

Ex: {t.FN / Stud(t) ∧ t.M > 50}

3) Bounded variables: var with quantifiers (∃, ∀)

4) Free variables: var without quantifiers

5) {t / t}
var that occur on left side & free var
var that don't occur on left side must be bounded on right side

6) ~∃t(P(t)) = ∀t(~P(t))
~∀t(P(t)) = ∃t(~P(t))
P → Q = ~P ∨ Q

1) Expressive power comparison:

Basic RA queries expressive power = Safe TRC queries exp power = Safe DRC queries exp power.

2) Basic RA queries :- queries which can express using π, σ, ×, ∪, −, ∩, ⋈, ⋈c, ⋈, ⋈, ⋈, /

3) Queries failed to express using Basic RA (Safe TRC) :-

- Count of records / attribute values.
- Sum of attr values or average.
- order in Asc/Desc order of attr value
- Representation of duplicate records in result.
- Insert / Delete / updation of records

NOTE: If length of query not constant, it not possible to express using RA

DRC:

- Syntax: {x1, x2, .. | P(x1, x2, .. xn)}
(result: what we want; condition: we need to declare var for all attr of all table under consideration)

Ex: {U, V | (∃U)(∃V)(Emp(U,V)) ∧ U='John'}