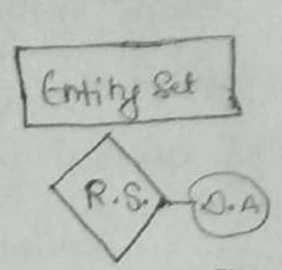
DBMS:

• Conceptual DB design (using ER MODEL): (nouns)



① Entity Set → Collection of similar entities.

② Relationship Set → Set of all similar relationships

③ Attribute → Belongs to E.S/R.S.

(i) Simple Attribute: cannot be divided further.

(ii) Single Valued Attributes: atmost 1 value.

(iii) Stored Attribute: Whose value need not be changed.

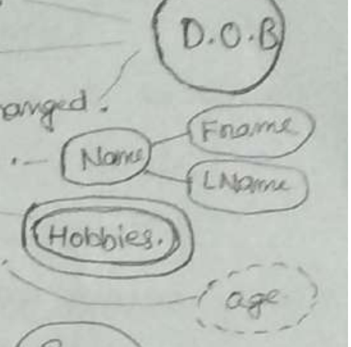
(iv) Composite attribute: Can be divided further.

(v) Multi-valued attribute: more than 1 value.

(vi) Derived Attribute: derived from other attribute.

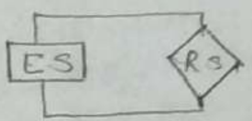
(vii) Key Attribute: used for unique identification of each entity set.

(viii) Discriptive Attribute: describes a Rel. Set.

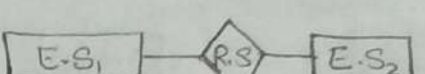


→ Degree of Relationship: No. of E.S participating in R.S.

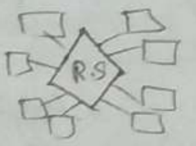
(i) Unary Relationship: → (Recursive Relationship)



(ii) Binary Relationship:



(iii) N-ary Relationship: →



→ Key Constraint: For an entity in the entity set, there is atmost one relationship in the relationship set.

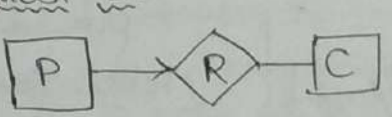
denoted by arrow

→ Participation Constraint:

↳ Total Participation: If every entity in the E.S participates in R.S else Partial Participation.

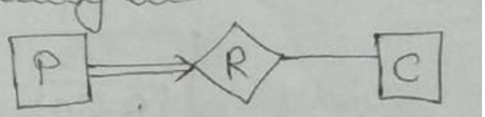
denoted by thick line or double line.

(i) atmost 1:



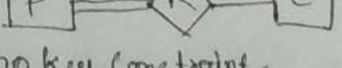
max = 1 = Key Constrain.
min = 0 = Partial participation.

(ii) Exactly one:



max = 1 = Key Constraints.
min = 1 = Total participation.

(iii) atleast 1:



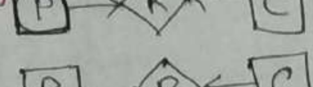
max = n = no key Constraint.
min = 1 = Total participation.

(iv) 0 or Many:

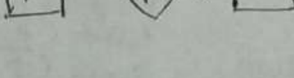
max = m = no key
min = 0 = Partial.

→ Cardinality Ratios:

(i) 1:1

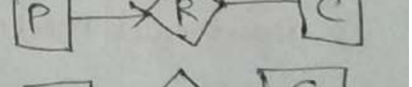


(ii) 1:m

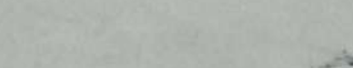


Every entity in C is related to atmost one entity in P.

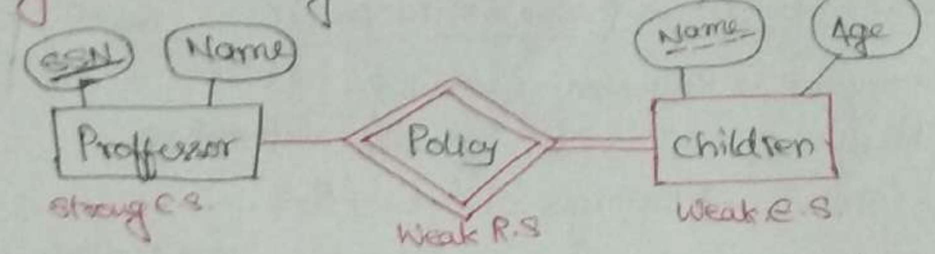
(iii) m:1



(iv) m:n



- Strong Entity Set : That has a key attribute
- Weak Entity Set : No key attribute.



Partial key or discriminator

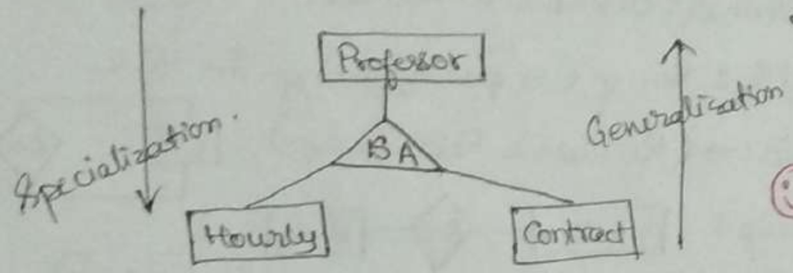
NOTE: ① Weak E.S cannot exist without depending upon Strong E

② The R.S between a strong & a weak E.S. is always wea

③ The participation from weak E.S to R.S is always Total

④ Each Weak E.S has exactly one owner.

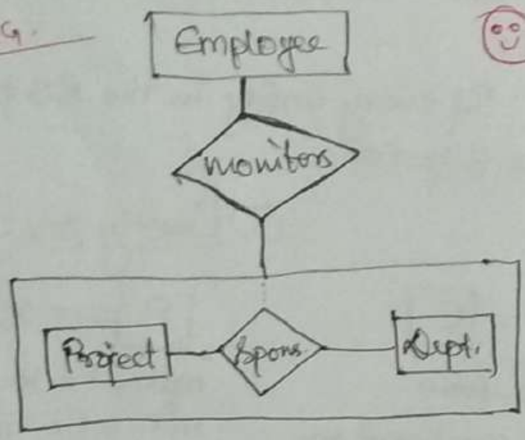
- CLASS HEIRARCHY : To show relationship b/w Generalized & Specialized E.
  ↳ Denoted by ISA Relationship.



- Some prof. are either hourly paid or contract based but not both. Thus, lots of space wastage.
- ☺ We use class heirarchy, for efficient space management.

- AGGREGATION : It is a relationship set that participates in another R.S. (a.k.a Relationship over a Relationship).

a.k.a. MODELLING.



☺ used as effective Communication tool b/w designer & customer as it is easy to understand.

☹ Every requirement cannot be represented diagramatically hence some dataloss may occur

NOTE: If we have key Const. & Total participation from both sides then it can be represented as one relation.

In Relational Model.

(i) Class Heirarchy.

Professor (<u>SSN</u>, Name)
Hourly (<u>SSN</u>, Hid, HP)
Contract (<u>SSN</u>, Cid, duration).

(ii) Aggregation.

Employee (<u>Eno</u>, Ename)
Projects (<u>Pno</u>, Pname)
Dept (<u>Dno</u>, Dname)
Spons (<u>P.no</u>, <u>D.no</u>)
Monitors (<u>Eno</u>, <u>Pno</u>, <u>Dno</u>)

- LOGICAL DATABASE DESIGN (using Relational Model):

Relation: defined using Relational Schema (structure) and Relational Instance (tuples present in a relation at any instance of time)

- Degree of a Relation : No. of attr. in the relation.
- Cardinality of a Relation : No. of tuples in the relation.

> Integrity Constraints : Condition/restriction on a tuple to be stored.

Table Integrity Const [① Primary key ② Unique key ③ Not Null ④ Check]

⑤ Foreign key. ← Referential Integrity Constraints.

> Key Constraints: Key → an attribute for unique identification of tuples.

Candidate keys: minimal set of attributes.

Superkey Contains a C.K.

Primary key One of the CK is chosen.

> Foreign key Constraints: → Can contain duplicates/Null value.

→ Every value present in FK of referencing relation (child) must also be present in the PK of referenced relation (parent)

→ No explicit insertion can be done on referencing relation, " " " referenced " . and No " deletion can be " "

- ON DELETE CASCADE: If whenever a tuple is deleted from the referenced relation, all the related tuples in the referencing relation ~~are~~ is also deleted.
- ON UPDATE CASCADE: (Similar for update)

ON DELETE or ON UPDATE
⇓
(i) Cascade
(ii) Set Null
(iii) Set default.

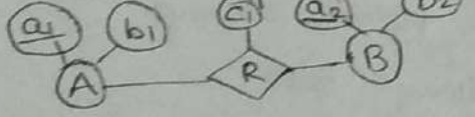
ER to Relational Model Conversion:

(i) Every E.S represented using one Relation.

(ii) Attribute of Relation includes only simple attribute of E.S ; Key attribute becomes P.K of the relation.

(iii) If multivalued attribute in E.S, then the relation is decomposed into multiple relation.

- Relationship Set:



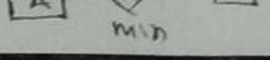
∴ A($\underline{a_1}$, b_1)
B($\underline{a_2}$, b_2)
R($\underline{a_1, a_2}$, c_1).

- CARDINALITY RATIOS:

① Many to Many.

A($\underline{a_1}$, a_2)
B($\underline{b_1}$, b_2)
R($\underline{a_1, b_1}$)

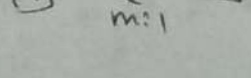
∴ 3 tuples.



② M to 1.

AR($\underline{a_1}$, a_2, $\underline{b_1}$)
B($\underline{b_1}$, b_2)

∴ 2 tuples.



③ 1 to Many.

A($\underline{a_1}$, a_2)
BR($\underline{a_1, b_1}$, b_2)

∴ 2 tuples.

④ 1 to 1,

Either AR & B
or
A and BR.

∴ 2 tuples.

SCHEMA REFINEMENT: • Problems due to Redundancy → Repeated Storage
↳ Insertion, Updation & deletion problems.

- To analyze and (solve) correct the problem caused due to redundancy, we use Normalization.

- X → Y is a Trivial FD, iff $Y \subseteq X$ or $Y = X$.

- Application of attribute Closures:
  (i) To find additional FDs
  (ii) To find equivalent FDS: F = G, iff F covers G & G covers F. (If every dependency of F is in closure of G.)
  (iii) Minimal Set of FDs: (a.ka Canonical/Irreducible set/Minimal cover)
  RULE 1: If X → Y holds and XY → Z exists, then delete Y.
  ∴ Final dependency. X → Y and X → Z.
  RULE 2: Remove transitive Redundancy (X → Y, Y → Z, ~~X → Z~~)
  (iv) Finding keys of a relation: Let R(ABCD)

Single CK. → CASE 1: CK = AB, then $\boxed{\#S.K = 2^{|ABCD| - |AB|}} = 2^2 = 4$.

Multiple CK. → CASE 2: CK = AB, BC, then $\boxed{\#Sk = 2^{|AB|} + 2^{|BC|} - 2^{|B|}}$ common.
$= 2^2 + 2^2 - 2^1 = 6$.

Similarly: CK = AB, C,
then $\#Sk = 2^{|AB|} + 2^{|C|} - 2^{|\emptyset|} = 2^2 + 2^1 - 2^0 = 5$

DECOMPOSITION:
(i) Lossy Decomposition: $[R_1 \cap R_2 = \emptyset]$ If R is decomposed into R_1 & R_2 and Join of R_1 & R_2 produces sporious (extra) tuples.
(ii) Lossless Decomposition: Join of R_1 & R_2 results into R again and the common attr. of R_1 & R_2 must be a key in either of R_1 or R_2 or both. $[R_1 \cap R_2 \neq \emptyset]$
(iii) Dependency Preserving: If every dependency in R can be determined using decomposed relation.

Good decomposition

NORMALIZATION:
(i) 1NF: Only Atomic Value (By default in Relational Model).
(ii) 2NF: Disallows PFD (Prime attribute ↛ Non-Prime attribute).
(iii) 3NF: Disallows Transitive Dependency (SK → Any or Any → Prime attr.).
(iv) BCNF: Disallows Prime transitivity. → (Only SK → Any).

NOTE: • BCNF may lose some dependency, Hence Dependency Preserv. decomp. into BCNF may not be possible always.
- A relation with 2 Attr. → Always BCNF
- If all Attr. of 'R' are prime Attribute → Always 3NF.
- Relation containg a key with 1 attr → Always 2NF.
- If R is in XNF, then its decomposed relation can be in ≥XNF (never <XNF).

RELATIONAL ALGEBRA: → (duplicates are eliminated implicitly)

Procedural Language (represents a process)

① Selection Operators → (σ) → To select rows.

② Projection Operator → (π) → To select columns.

③ Set Manipulation Operators → ($\cup, \cap, -$) → B/w Two Compatible Relation.

same no. of attr having same domain names.

④ Cartesian Product → If R has 'm' rows and 'a' attributes and S has 'n' rows and 'b' attributes, then R×S has m×n rows and a+b attributes.

⑤ JOIN ($\bowtie_c$): Cartesian Product followed by selection, then projection.

$$R \bowtie_{R.B=S.B} S = \pi_{A,B,C}\left[\sigma_{R.B=S.B}(R \times S)\right]$$

No. of tuples
min = 0
max = m×n

⑥ Natural JOIN ($\bowtie$): Join without Condition (By default, the condition is equality on all common attributes only).

⑦ Rename Operator (ρ): used to represent the result of R.A. Expr. with symbolic name.

(i) ρ(A, Rel. Algebra Expr.) → result renaming.

(ii) $\rho_{x:1}(R)$ → Column renaming (Here, rename 1st Column).

⑧ Division Operator ($\div$): To compare one with all.

$A(x,y,z) \div B(z) = (x,y)$ for all z.
$A(x,y,z) \div B(y,z) = x$ for all y and z.
$A(x,y) \div B(z)$ = invalid (z is not present in A).

RELATIONAL CALCULAS: describes the result (not the process) → Declarative Language (non-procedural).

(i) Tuple Relational Calculas: Result is described as set of tuples of a relational schema (Tuple Variables).

→ $\{T \mid P(T)\}$
formula that describes T.
Tuple variable.

(ii) Domain Relational Calculas: Result is described as set of values of an attribute's domain (Domain Variables).

→ $\{\langle x_1, x_2 \cdots x_n\rangle \mid P(x_1, x_2 \cdots x_n)\}$
(Domain Variables) (formula that describes $\langle x_1, x_2 \cdots x_n\rangle$)

- UNSAFE QUERY: A query that returns infinite no. of rows as a result

Eg: $\{T \mid \neg T \in R\}$ → unsafe

STRUCTURED QUERY LANGUAGE:

- ORDER OF EVALUATION: ① FROM (Cartesian product of all tables).
  ② where (to select rows based on some condition).
  ③ Group by (divide the rows into Groups).
  ④ Having (Select the groups)
  ⑤ Mathematical Expressions.
  ⑥ Distinct (elimination of duplicates).
  ⑦ Order by (Sort the rows).

- CONNECTIVES:

- and : ... where A='x' and B='y'.
- or : ... where A='x' or B='y'.
- not : ... where not A='x'.
- (not in) in : ... where A in ('x','y','z')
- (not btw) Between : ..... where A between 10 and 20. inclusive.
- (is not null) Is NULL : ..... where A Is Null.
- like : ... where A like x% → starts with x.
  ... ——— %x → ends with x.
  ... ——— %x% → Contains x.
  .... where A like '___' → length 3 exactly.
  ... ——— 'x__%' → min 3 length & starting with x.

Null = 0 : False
Null <> 0 : False
Null ≠ 0 : False
Null = Null : False
Null Is Null : True

- NOTE: '%' → indicates 0 or more characters.
  '_' → indicates exactly one character.

- Set Manipulation Operations: $\boxed{\cap, \cup, -}$ on Compatible Relations
  ↳ Also, eliminates duplicates.

- Order By: → By default asc → If order by fails, then sorted in order of insertion.

  Eg: .... where .... order By A desc, B;

- Aggregation Functions: Takes only one set of values as input & returns a single value as output.

  ignores NULL (i) min (ii) max — number/ text; (iii) Sum (iv) Avg — Numbers only; (v) Count — no. of rows

  is a group/col^{m} fn thus it cannot be used with row fns like where & Order by

  NOTE: Count (Const. or *) → no. of rows.
  Count (Attr. name) → No. of valid values (excluding NULL)

- $\text{Avg}(A) = \frac{\text{sum}(A)}{\text{Count}(A)}$

- Agg. f^{ns} cannot be used in where or Order By.
- Agg. fns cannot be used with Attributes together (w/o a group by clause).

  e.g: Select Name, max(m$_1$) from Student; ✗ Error

• GroupBy & HAVING: → used to select the Groups.

GroupBy ↓ used to divide the rows into groups.

Eg: Select dept, city, count(*) from employee group by dept.

Let's assume it results in | R {Hyd, Pune} 3. | → violates 1NF

Thus, Select dept, city, count(*) from employee group by dept, city

R	Hyd	2.
R	Pune	1.

✓ NOTE: Every attribute in select must also appear in the GroupBy clause to avoid any violation.

NOTE: Having Clause contains only Agg. fns or an Attribute name that is being used in the Group by Clause.

- In the absence of GroupBy clause, the having clause considers the total relation as one group.
- If Grouping Attribute contains null, then all null forms a separate Group.

WHERE CLAUSE	HAVING CLAUSE
① used to select the rows	① used to select the Groups.
② Agg. fn cannot be used.	② Agg. fn can be used.

• JOIN: Select NAME from Professors join Teachers on Prof.SSN = Tchr.SSN

[OR] Select NAME from Professors Natural Join Teachers.

• SELF JOIN: It refers to Table joining by itself.

• Outer JOIN: (i) LOJ → returns all rows from left side (even if there is no matching row in the right side relation).

(ii) ROJ → returns all rows from right side relation.

(iii) Full outer JOIN → returns all rows from both relation

☹ Problems with JOIN: More time & Space Consuming.

☺ Nested Queries (Sub-Queries): Evaluated from Inner to Outer.

• SET COMPARISION OPERATORS: (i) in → ... where A in (x, y, z);

(ii) any(...) → ... where A > any (50, 60, 10);

(iii) all (...) → ... where A > all (50, 60, 10);

NOTE: all(Empty) = TRUE
any (Empty) = FALSE

- **WITH CLAUSE**: Provides a way of defining a temporary relation whose definition is available to that Query itself.

  e.g: With Max-Budget (Value) as (Select max(Budget) from Dept.)

  Select Dname from Department, Max-Budget where Budget = Value

temporary relation

Max-Budget

Value
5c.

Department.

Did	Dname	Budget
1	R	5Cr.
2	P	2Cr.

output: R

- CORELATED SUBQUERIES: Inner query depends upon Outer Query and also Outer Query depends upon Inner Query.
- EXISTS CLAUSE: used to compare with an Empty set.

** exists (!Empty) → TRUE.
exists (Empty) → FALSE.

TRANSACTIONS: Collection of operations performing a logical unit of work.

(i) Atomicity: Either all or no Operation of a transaction executes.
- Transaction Manager ensures Atomicity.

(ii) Consistency: State should be correct before/after the transaction.
- APPLication Programmer (User) ensures Consistency.

(iii) Isolation: Transaction assumes it is the only one running in the sys.
- Concurrency control Manager ensures Isolation.

(iv) Durability: All updates done on a transaction should be durable.
- Recovery Manager Ensures Durability.

SCHEDULES: Order of execution of transaction's Operations.

(i) Serial Schedule: Transactions are executed completely - One by One.
- It always ensures Consistency.
- For n transactions, n! serial Schedules are possible.

(ii) Concurrent Schedules: Context Switching of Transactions may occur.
- It is prone to inconsistency but it minimizes waiting Time, Response time. Also, CPU and I/O devices are utilized efficiently.
- For n transactions with $n_1, n_2, n_3 \ldots n_n$ operations,

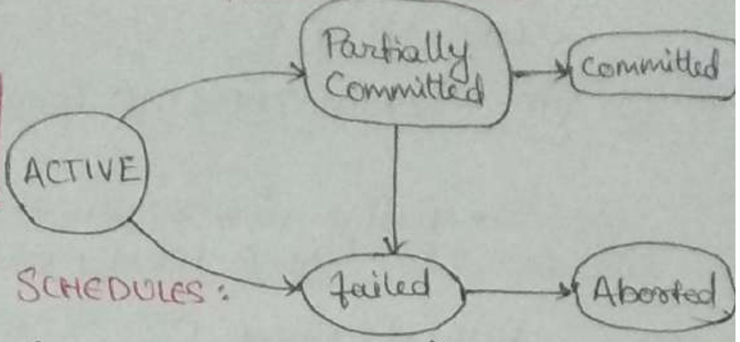
$$\frac{(n_1+n_2+\ldots+n_n)!}{n_1!\, n_2! \ldots n_n!}$$ Concurrent Schedules are possible.

- Concurrent Schedules may include Serial Schedules as well.

- Total no. of Non-Serial Schedules

$$= \boxed{\frac{(n_1+n_2+n_3+\cdots\cdots+n_n)!}{n_1!\,n_2!\cdots\cdots n_n!} - n!}$$

Transaction States.



PROBLEMS DUE TO CONCURRENT SCHEDULES:

(i) LOST – UPDATE PROBLEMS: → (write - write Conflict).
- Update of one transaction is over-written by Another transac

(ii) DIRTY - READ PROBLEMS: → (write - Read Conflict).
- Reading of uncommitted data.

(iii) UNREAPEATABLE - READ PROBLEM: → (Read - Write Conflict).
- A write of a trans. occurs between two reads of another trans

SCHEDULES BASED ON RECOVERABILITY:

(i) Non- Recoverable Schedule: T_j reads a value that was previousl written by T_i and T_j commits before T_i.
☹ Non- Recoverable and Cascading Rollbacks.

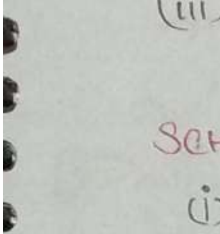
(ii) Recoverable Schedule: T_j reads a value previously written by T_i and T_j commits after T_i. ☹ Cascading Rollbacks & Recoverable.

No Dirty Read

(iii) CASCADING Rollbacks: A single transaction failures leads to abortion of multiple transactions.

(iv) Cascadeless Schedules: T_j reads a value previously written by T_i and T_i commits before the read operation of T_j.

(V) Strict Schedules: T_j can read/write the value returned by T_i only after T_i commits or aborts.
- No W-R or W-W conflict occurs here.



NOTE: Every Cascadeless Schedule $\xleftrightarrow{}$ Recoverable Schedule.

SERIALIZABILITY: Concurrent Schedule equivalent to Serial Schedule.

① CONFLICT SERIALIZABLE: Concurrent Schedule Conflict Equivalent to Serial Schedule.
Same Conflicts in both the Schedule.

∵ Serial Schedule is always consistent.
∴ Conflict Serializable Schedule is also Consistent.

Algo: (i) Draw a precedence graph and draw an edge from T_i to T_j in case of conflict.
(ii) If directed graph consists a cycle → Not Conflict Serializab
(iii) Order of serializability → topological sort where T_x wi no incoming edge is the first node & T_x with no outgoin edge = last node.

- Total No. of C.S. Schedule = Total Concurrent Sch. — Total N.C.S. Sch

② VIEW SERIALIZABILITY: Conn. Sch. view equivalent to Serial Schedule

View for each data item in both the Schedule are same.

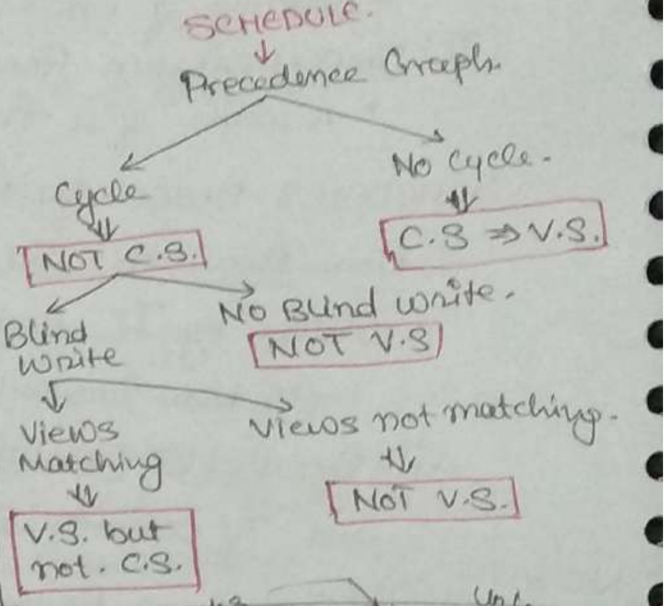
- Testing view serializability is N-P Hard problem.

NOTE: (i) Every C.S. Schedule → V.S. Schedule.

(ii) A V.S. but not C.S. schedule can only be possible when there is a blind write but vice versa not true

(iii) C.S. is more powerful than V.S.. Thus, C.S. is used practically.

Commits has no effect in finding C.S./V.S.



CONCURRENCY CONTROL: Sharing of data in efficient manner.

(i) LOCK BASED PROTOCOLS: Data items be accessed in mutually exclusive manner

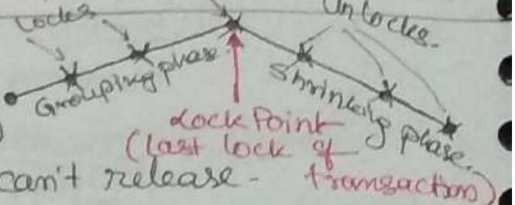
ⓐ Shared lock (LOCK-S) → Only Read.

ⓑ Exclusive lock (LOCK-X) → Read/Write.

	S	X
S	✓	×
X	×	×

- Two transaction can access a data item simultaneously iff both are using shared lock.

(ii) Two Phase Locking Protocols:



Phase I: Growing Phase (a.k.a. Locking Phase)

- Trans. Can acquire the locks but can't release.

Phase II: Shrinking Phase (a.k.a. Unlocking Phase)

- Trans. can release the locks but can't aquire.

NOTE: Any Schedule possible under 2PL (without deadlock), always ensures Conflict Serializability.

☹ • CR may occur under 2PL protocol.

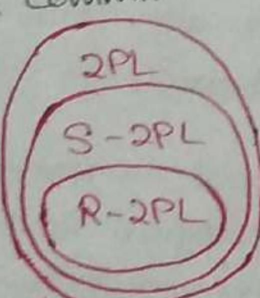
☺ • Unlock only after commit.

Deadlock Avoidance

ⓐ STRICT 2PL protocol:

→ All exclusive locks must be held until commit.

```
L-X(A)
  :
L-S(B)
  :
U(B)
Commit
U(A)
```



ⓑ Regorous 2PL protocol

→ All locks must be held until Commit.

```
  :
Commit
U(A)
U(B)
```

☹ Deadlocks may occur under 2PL.

DEADLOCK HANDLING TECHNIQUES:

(i) Deadlock Prevention : (needed only when prob. of deadlock is high)

ⓐ Wait - die Protocol : (GandhiGiri) →
older process waits and younger processes aborts willingly.
• To avoid Starvation to younger process, it is restarted with the same timestamp.

ⓑ Wound - Wait Protocol : (GundaGiri)
Younger processes waits but older process wounds and makes the younger process to abort forcefully.

(ii) Deadlock Detection : Draw a wait-for-graph, if cycle exists then deadlock, else, no deadlock.
• For Recovery, choose one of the transaction as victim & kill.

(Size of data units)
(iii) MULTIPLE GRANULARITY: Heirarchical representation in the form of tree. It is also a variant of 2PL (hence, Deadlock may Occur).
• Locking → from Root to leaves • Unlocking → from leaves to root.
• Deadlock Avoidance → Multiple Granularity Strict/Regorous 2PL.
→ This protocol is implemented using Additional lock mode

IS → (i) Intention Shared : It indicates that Shared S lock will be requested on some descendent nodes.

IX → (ii) Intention Exclusive : Exclusive X locks will be requested on some descendent nodes.

SIX → (iii) Shared and Intention Exclusive : Current Node is locked in Shared S mode and Exclusive X locks will be requested on some descendent nodes by the same transaction.

NOTE! • If a node in a DB is locked using some ~~Intention~~ lock mode (S/X), then its all children is also locked using the same mode [except for IS, XS, SIX].
• If a node in a DB is locked using SIX mode, then other transaction can only using IS mode (not even shared mode as one child is locked in X mode).

	IS	IX	S	SIX	X
IS	✓	✓	✓	✓	X
IX	✓	✓	X	X	X
S	✓	X	✓	X	X
SIX	✓	X	X	X	X
X	X	X	X	X	X

⇐ Compatibility Matrix.

(iv) TIMESTAMP BASED PROTOCOLS: Order of Serializability is determined in the order of their timestamps.

- If $T_S(T_i) < T_S(T_j)$, then the resultant schedule must be equivalent to serial schedule T_i to T_j.

ⓐ Timestamp ORDERING Protocol (TOP): all conflict Operations must be executed in the order of their time stamp.

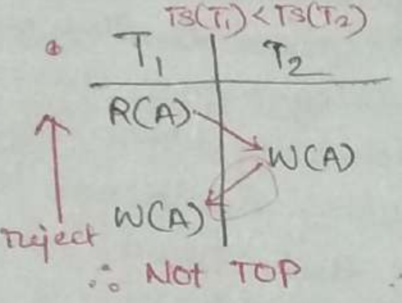
→ If a transaction gets involved in conflict not executing in the order of Timestamp, then it is aborted and restarted after sometime with a new timestamp.

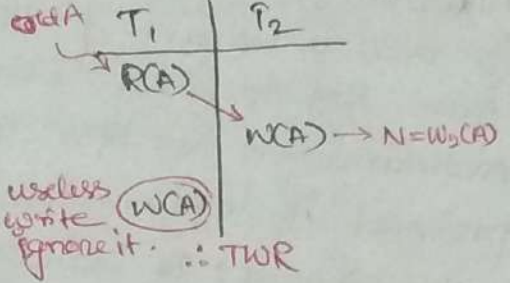
☹ Starvation may occur due to continuous abort & restart of a transaction.

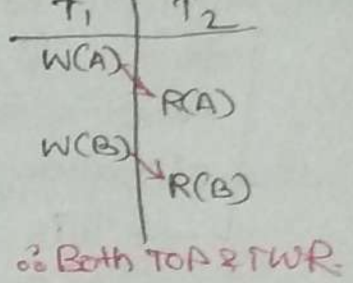
- It ensures Conflict Serializability.

ⓑ Thomas - Write Rule (TWR): useless writes are ignored.

- It ensures View Serializability. ☺ No Starvation.







NOTE: Every TOP $\overset{\not\leftarrow}{\rightarrow}$ TWR.

FILE ORGANIZATION & INDEXING:

DB
↓
Files
↓
Blocks
↓
Records
↓
Data.

- Block Access: Transfer time of a block from disk to memory.
- Blocking Factor: Avg. no. of records that can be stored in a Block.

$$\boxed{BF = \frac{\text{Block Size.}}{\text{Record Size.}}}$$

- For storing records into a block, we have two techniques.

(i) SPANNED STRATEGY:

- Allows partial part of record to be stored in a block.

☹ More no. of Blocks to be accessed for accessing a Record.

☺ No wastage of Memory.

- Suitable for Variable length records.

(ii) UNSPANNED STRATEGY:

- Only Complete records can be stored in a Block.

☹ Wastage of Memory.

☺ Less no. of Blocks to be accessed for accessing a record.

- Suitable for fixed length records.

For storing records in a file, we have two Approaches:

① ORDERED FILE ORGANIZATION:

- Records are stored in the order of Some attribute (usually P.k)
- It uses Binary search.
- ☹ Insertion is expensive.
- ☺ Searching is efficient.

② UNORDERED FILE ORGANIZATION.

- Any record can be stored at anyplace in the file (usually at the end of file).
- It uses Linear Search.
- ☹ Searching is expensive.
- ☺ Insertion is easy.

INDEX: It is an ordered file with 2 fields.

Index.	
key	ptr.

on which Binary search is performed.

(i) Single Level Index: Binary Search

ⓐ Primary Index (Pk+ord.)

- Sparse Indexing
- Avg. no. of Block Access to search a record = $\lceil \log_2 B_i \rceil + 1$

Block Anchor →

P.k	Block Pointer.

- Index record is created for the 1st record of each Block
- No. of index Records = No. of Blocks in data-file

ⓑ Clustered Index (NK+ord)

- Sparse Indexing.
- Avg. no. of ...... = $\lceil \log_2 \underset{\uparrow \text{Index Blocks.}}{B_i} \rceil + 1$

N.K	Block Pointer

- Index record is created for each cluster.
- No. of Index Records = No. of Clusters = No. of distinct NK values

ⓒ Secondary Index ($\begin{smallmatrix}\text{NK}\\\text{or}\\\text{CK}\end{smallmatrix}$ + $\underset{\text{Linear Search}}{\text{Unord}}$

- Dense Indexing.
- Avg. no. of Block access ...... = $\lceil \log_2 B_i \rceil + 1$

N.k or C.K.	~~Block~~ Record Pointer

- Index record is created for each record in datafile.
- No. of Index Records = No. of records in datafile
- It provides logical ordering of a datafile.

NOTE! All three (P.I, C.I. and S.I) are ordered on which Binary Search is perform

uses Block pointers ⇒ • Sparse INDEXING: It contains Index Records only for some Records (Blocks) in DF

uses Record pointers ⇒ • Dense INDEXING: It contains " " for every Records in Data File

(ii) Multi-Level Index: Implemented using B-trees and B^+-trees.

B-trees.

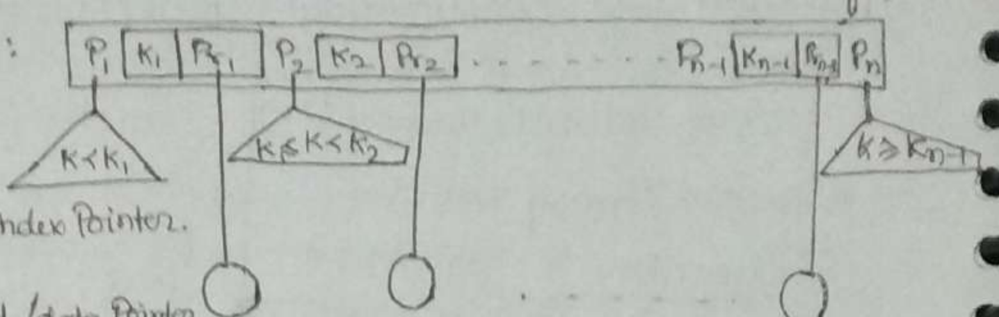
(i) Data is stored in leaf as well as in internal nodes.
(ii) Searching is slower.
(iii) No Redundant Search key.
(iv) Deletion is Complex.
(v) Leaf nodes cannot be linked together.

B^+-trees.

(i) Data is stored in leaf nodes only.
(ii) Searching is faster.
(iii) Redundant search keys may be prese
(iv) Deletion is easy (because of point(i)).
(v) Linked list of leaf nodes is present here.

- B-trees → It is a self-Height-Balanced tree (all leaves on the Same levels).
  → Each Block in a multilevel index is Represented as a Node of a B-Tree

- Structure of a NODE:


P_1 | K_1 | Pr_1 | P_2 | K_2 | Pr_2 | ...... P_{n-1} | K_{n-1} | Pr_{n-1} | P_n
$K < K_1$
$K_1 \leq K < K_2$
$K \geq K_{n-1}$

Here,

ORder of B-tree ='n'
- $P_1, P_2 \ldots P_n$ → Block/Tree/Index Pointer.
- $K_1 < K_2 \ldots < K_{n-1}$ → Keys.
- $Pr_1, Pr_2, \ldots Pr_{n-1}$ → record/data Pointer.

$$\boxed{\therefore (n \times P) + (n-1)(K + Pr) \leq \text{Block Size}.}$$

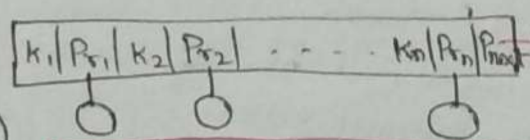
- B-tree Insertion: Insertion of key values is always done at the leaf Node. If a node is full, then split it into two nodes.
- The middle element Becomes root node, elements less than middle goes to left-Substree.
- For order of B-tree ='n'; max: $\overset{\text{Block/tree/Index pointers}}{np}$, $\overset{\text{Keys}}{(n-1)K}$
  min: $\lceil \frac{n}{2} \rceil P$, $(\lceil \frac{n}{2} \rceil - 1)K$

- B-Tree Deletion:
  1. If the key is in leaf node, then deletion causes no problem.
  2. If key is not a leaf node → internal node, then, ~~no. of keys in the node [illegible] is present~~ min. key
     - (a) If the child that precedes key has at least 't' keys, then replace the key with it's inorder predecessor.
     - (b) If the child that precedes key has fewer than 't' keys, then, check the child that succeeds the key.
       - (i) If the child that succeeds the key has atleast t keys, then replace the key with its inorder Successor.
       - (ii) If the child that ...... has fewer than t keys, then merge the preceder & successor child into 1 node.
  3. (a) If the root has fewer than t keys, ...

	BP	KP
max:	n	$n-1$
min:	$\lceil \frac{n}{2} \rceil$	$\lceil \frac{n}{2} \rceil - 1$

↑

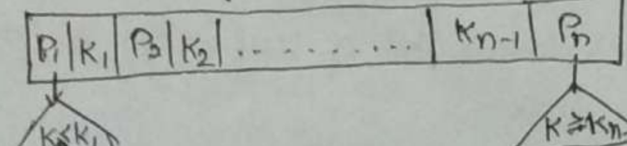
- B^+-trees: • Structure of LEAF NODE:


K_1 | Pr_1 | K_2 | Pr_2 | .... K_n | Pr_n | P_{next}
→ to sibling node.

- Order of leaf node ='n' (max. no. of data)

$$\boxed{\therefore n \times (K + Pr) + P_{next} \leq \text{Block Size}.}$$

- Structure of INTERNAL NODE:

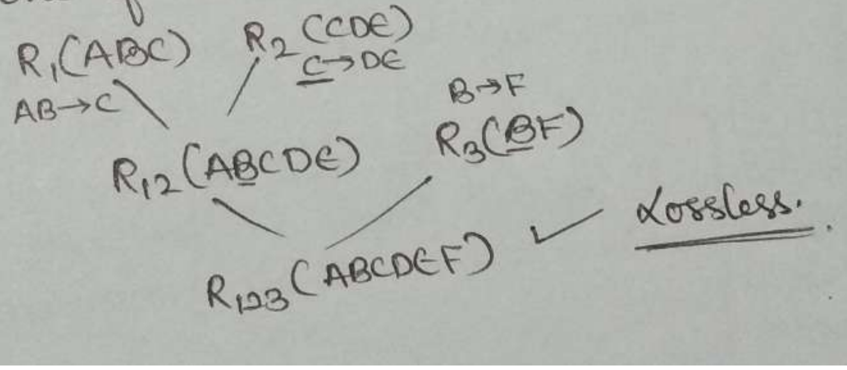

P_1 | K_1 | P_2 | K_2 | .... | K_{n-1} | P_n
$K \leq K_1$
$K \geq K_{n-1}$

- Order of internal node = n. (max. no. of tree ptrs).

$$\boxed{n * p + (n-1)K \leq \text{Block Size}.}$$

Points:

1. Row Level of locking provides Higher Degree of Concurrency with exception of root.
2. Empty Relation never violates any FDS.
3. Space utilization of B+ tree index = 50% (min) to 100% (max).
4. If Isolation level in transaction is "read-committed", then, Dirty reads are not allowed but unrepeatable reads are allowed.
5. If $R_1(ABC)$, $R_2(CDE)$, $R_3(BF)$ is decomposed from 'R', Then try to combine any two relation such that there is a common element between them and the common element is a key in any one of the two relation.



R1(ABC)
AB→C
R2(CDE)
C→DE
B→F
R12(ABCDE)
R3(BF)
R123(ABCDEF)
✓
Lossless.