Module-1 Relational model & Normalisation

PAGE NO. :
DATE : 21 / 11 / 2023

DBMS (Not In short)

★★ Overview

→ Database is (huge) interrelated data & DBMS is sfw package that manges it. (interface b/w user & DB).

→ Database System = DBMS + DB
↳ sfw having programs.

★ History:

1960's : n/w model (graphh)
1970's : using tree model (hierachical)
1980's : Relational model given by EF Codd
↳ Till now

→ Despite having FS in OS we use DBMS bcz it has:
• Concurrency • Less Redundancy • No inconsistency

→ Advantages:
• Data independence i.e. provides abstract view to application
• Efficient data access using sophisticated techniques
• Data integrity & security
• Concurrent access & crash recovery.

→ Design process:

Req. analysis → Conceptual Design (ER model) → Logical Design (Relational model) → Imp

→ Model is nothing, just set of rules & conventions to repr. the information. We use it bcz it's made w/o any cost & provide good analysis
Ex: Earth is modelled as globe / maps.

PAGE NO :
DATE : / /

★ Relational Model:

→ In this Table = Relation
Row = Tuple / Record
Column = Attribute / Field

→ Relation has schema and particular instance (like ss of it).
↳ Table name, Attribute name & domain

→ Every attribute has a domain
↳ set of atomic values
↳ Indivisible

→ Degree of Relation = Arity of Relation = # Attributes in it
→ Cardinality of Relation = # Rows in any instance of relation

→ Table is a relation as it's subset of domain's cross product.

In maths relation, $R: A \to B$ is $R \subseteq A \times B$
Here, if domain is $D_1, D_2 \ldots D_n$ of attributes then relation is subset of $D_1 \times D_2 \times D_3 \ldots \times D_n$

→ Every row $r \in D_1 \times D_2 \ldots \times D_n$

→ Instance is just set of records/rows.

→ formally, relation is a subset of a cartesian product of sets.

PAGE NO. :
DATE : / /

Qu: The Relation R's schema is:

$R(A_1 : D_1, A_2 : D_2, A_3 : D_3)$

and $|D_1| = 2$, $|D_2| = 3$, $|D_3| = 4$

then

a) Cardinality of largest instance is?

⇒ $2 \times 3 \times 4 \Rightarrow 24$

b) Instances possible are?

⇒ 2^{24} as all 24 rows have 2 choices.

→ Attribute values – are atomic
– have a known domain
– Can be "null" sometimes

→ Three meaning of null values:

a) Not applicable, Ex: wife's name for karan

b) Not known, Ex: DOB sometimes

c) Not recorded, absent

→ As table is set of records, so, order of tuples do not matter, but order of columns matter as tuples/rows are ordered pairs.
And, no duplicate rows in relation as it's set of rows.

PAGE NO. :
DATE : / /

★★ Keys:

→ **Superkey**: A set (can be singleton also) of attributes that can identify any record uniquely. i.e. no two records can have same values (non-null) for these attributes

- In worst case, atleast set of all attributes is a superkey bcz. it will be unique.
- Super set of superkey is superkey.
- Null values are allowed in it.

→ **Candidate key**: Minimal Superkey

If $CK = A_1A_2 \ldots A_n$ then

$\forall i$ $(CK - A_i)$ is not a superkey.

→ **Primary key**: One of the CK is selected to be primary key which must be not null.

Ex:

Col A	Col B	Col C	Col d	Cole	Col f
A	A	A	1	a	1
B	A	X	2	a	2
C	X	Y	3	x	1
D	X	Null	4	y	2
E	Y	2	5	2	1

Simple ← C.K = {colA, colC, Cold} {Cole, colf} → Composite CK.

P.K can be colA or Cold but not col_c due to null.

or {Cole, colf}

PAGE NO.:
DATE: / /

→ At any point of time, we have ~~atmost~~ exactly one PK for each relation. (theoritically, in Rltnal model)

→ Simple CK have 1 attribute and composite ck have ≥2 attr.

→ Collection of instance of each relation of DB is called snapshot of DB.

→ Codd gave Relational model and then also gave two mathematical & less practical language for data retrieval:
- Rltnal algebra
- Rltnal Calculus.

→ SQL is a professional language with updates each year based on relational model made by IBM.

Ques: Which of the following can be set of CKs

(a) {a,bc} (b) {} (c) {abcde} (d) {a,b,c}
(e) {ab,bc} (f) {a,ab}
can't be together both.

Ques: In R(a,b,c,d,e) if abcde is CK then # SKs?
⇒ 1 only. as no superset & subset possible

Ques: In R(a,b,c,d,e) max. no. of SK's possible.
⇒ $2^5 - 1$ if {a,b,c,d,e} are CKs
↳ Powerset ↳ ∅

Ques: max. # CKs possible for following relations:

PAGE NO. :
DATE : / /

${}^{n}C_{k}$ is max when $k = n/2$

a) R(a,b,c)

→ 3 {a,b,c} as {ab, bc, ca}

b) R(a,b,c,d)

→ 6 {ab, bc, cd, da, ac, bd}
i.e. ${}^{4}C_{1}$
if take of len=1 then 4, and for len=3, then 4 i.e. ${}^{4}C_{3}$

c) R(a,b,...,n)

→ ${}^{n}C_{n/2}$ as of size $n/2$ are max.

→ So in R(a, a2 ... an) max #CKs can be ${}^{n}C_{n/2}$ and max. #SKs can be $2^{n}-1$.

Ques: In R(a,b,c,d,e) CKs = {a,bc} , #SKs?

⇒ $2^{4} + 2^{3} - 2^{2}$ ⇒ 20 (Inc. Exc. principle)

2^{4} → Super set of a
2^{3} → Superset of bc
2^{2} → Superset of abc

Ques: In R(a,b,c,d,e) CKs = {ab, bc, cd} then #SKs?

⇒ By inc exc. principle.

$2^{3} + 2^{3} + 2^{3} - 2^{2} - 2^{2} - 2^{1} + 2^{1}$

Superset of: ab bc cd abc bcd abcd abcd

= 16

PAGE NO. :
DATE : / /

→ As SQL is a practical lang. loosely based on Rel. alg. & calculus it gives us some flexibility such as instead of set relation is multiset of row i.e. duplicate rows are allowed & also PK is not mandatory.

★ **Integrity Constraints over Relations:** A database is only as good as the info. stored in it & a DBMS must therefore help prevent the entry of wrong info. in it.

An integrity constraint is a condition that is specified on a database schema. If instance satisfies all IC's it's a valid instance and DBMS enforces IC, in that it permits only valid/legal instance to be stored in database.

a) **Domain Integrity Constraint:** On a particular attribute. → CHECK kw of SQL (Range of values can also be given)
Ex: S_id must be integer.

b) **Key IC:** On any CK (Uniqueness IC) → UNIQUE kw of SQL
Ex: S_id must be Unique

c) **Entity IC:** On PK (PK IC), Not null & unique → PRIMARY KEY / NOT NULL UNIQUE kw of SQL
Ex: S-id must be unique & mustn't be null.

d) **Functional Dependency IC:** B/w two set of attributes of same / two not imp. directly.
Does reduces incons. of FDs.
(More details ahead in Module)

e) **Referential IC:** (FK IC) b/w two tables or within a table On foreign key. → REFRENCES kw of SQL
(Now next)

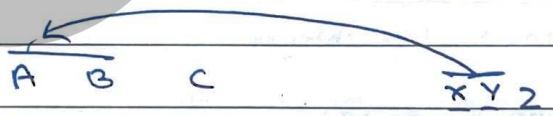
PAGE NO. :
DATE : / /

☆ Foreign key and Refrential Ic:

Ex: Student (S_id, name, c_id)
Course (cc_id, name, instructor)

Course.c_id & student.c_id are cand. keys and Student.c_id is a foreign key and student rltn is refrencing rltn and course is refrenced relation.

→ fk references to a candidate key
→ fk itself need not be candidate key i.e. need not be unique.
→ FK may or may not be allowed to be NULL, by default allowed to be null.
→ FK can be made up of more than one attr. also
Ex:



→ Fk can point to CK of same table also.

*→ Refrential Ic say that if R.X is ref. S.Y then R.X can be either null or can be from S.Y.

It can be violated in following conditions:

	In. Referencing Table	In Refrenced Table
Row Deletion	X	✓
Row Insertion	✓	X
Row updation	✓	✓

→ If RIc violation is in refrenced relation then we can:
a) Cascade the operation
b) Set null
c) Set default value
d) Ignore / Restrict / No Action SQL keyword

PAGE NO. :
DATE : / /

→ If violation happens in referencing table then that violating opⁿ is rejected/not allowed.

→ In fact any IC violatin opⁿ is rejected, except ones due to deletion of some tuple which is only ref IC in referenced table.

★★ Functional Dependencies: These are another type of IC that may be responsible for redundancies in the relation.

These are similar to "implication" from prop. logic.

Ex: City → State ; F.D.

i.e. city determines state.

then City State.

Jpr → Raj.

Jpr → Raj

jpr → up → Not allowed. bcz city → state.

→ Let A, B be two attributes in a relation R then

A → B iff

$(t_1.A = t_2.A) \rightarrow (t_1.B = t_2.B)$

for any two tuples of R.

→ FD's, PK's, FK's etc. are all decided by DB owners.

→ FD's are only one way:

city → State

→ A → B means if we know value of A then uniquely we know B.

PAGE NO :
DATE : / /

→ FD can also be defined blw set of attributes like

Ex: $\{A,B\} \rightarrow \{C,D\}$

A	B	C	D	E
2	2	1	2	1
2	3	3	4	2
2	2	1	2	2
2	3	3	4	1
1	3	3	4	1

→ There is abuse of notation here:

$\{A,B\} \rightarrow \{C\}$ is written as $AB \rightarrow C$ for our conv.

→ $X \rightarrow X$ is a trivial FD as, if I know X then I can say uniquely X.

NOTE ⟷ FD is defined on schema, not on instance, that's why it must be satisfied by all of the instances.

By seeing a instance we can't say which FD hold but we can say surely which do not hold.

Ex: For below instance.

A	B	C
2	2	1
3	3	2
4	4	3

$A \rightarrow B$ is true for instance but can't say anything about schema. and $A \rightarrow C$ do not hold on schema.

PAGE NO. :
DATE : / /

Qu: Consider the below instance and comment on FD's on schema:

A	B	C	D
1	1	1	1
1	2	2	2
2	2	2	3
3	3	4	3

a) $A \to B$ do not hold.

b) $A \to C$ " " "

c) $B \to D$ may or may not hold.

d) $A \to A$ definitely holds (Trivial)

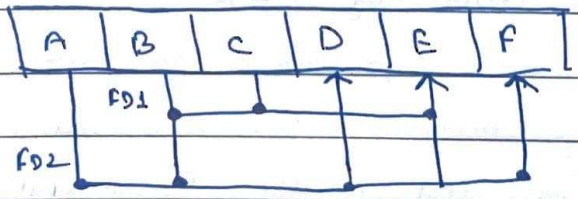
e) $AB \to C$ may or may not hold.

f) $A \to BC$ " " " " "

Qu: Consider $R(A, B, C, D, E)$ with FDs: $AB \to C$ and $CD \to E$. If atmost 3 diff values for A, B, D, what's the max. no. of diff values for E.

⇒ At most 9 diff value of C from $AB \to C$ and atmost 27 diff values of E from $CD \to E$.

→ Some author may repr. FD's as



Diagrammatic notation of FDs

FD_1 : $BC \to F$

FD_2 : $AB \to DFE$

	Split	Combin
LHS	✗	✓
RHS	✓	✓

PAGE NO :
DATE : / /

→ For gate we don't consider null values while discussing FDs Lien published NFDs in Research paper. (Null FDs)

⇒ Rules for FD's.

a) **Transitive Rule:**

$A \to B$ & $B \to C$ then $A \to C$

b) **Splitting Rule:**

if $ABCD \to XY_2$

then $ABCD \to X$, $ABCD \to Y$, $ABCD \to Y_2$

but $ABCD \to XY_2$ ↳ might be false.

c) **Combining Rule:**

if $ABC \to X$ and $ABC \to Y$ and $D \to X$

then $ABC \to XY$ and $ABCD \to X$

So, it works on both LHS & RHS

d) **Trivial FD's:**

$X \to X$, $XY \to X$,

$X \to XY$ is not trivial.

$X \to Y$ is trivial if $Y \subset X$.

→ Remember these are beating like A can beat B & C can beat B then AC can beat B, but AB can beat C then can't say alone A can beat C or not.

Analogy:

PAGE NO. :
DATE : / /

All rules at glance:

- Reflexivity — if $x \supseteq y$ then $x \to y$
- Augmentation — if $x \to y$ then $xz \to yz$
- Transitivity — if $x \to y$ & $y \to z$ then $x \to z$

} Armstrong's axioms

- Union — if $x \to y$, $x \to z$ then $x \to yz$ (counter beating analogy)
- Decomposition — if $x \to yz$ then $x \to y$ and $x \to z$
- Pseudo transitivity — if $x \to y$ and $wy \to z$ then $wx \to z$
- Composition — if $x \to y$ & $z \to w$ then $xz \to yw$

all of the above properties can be easily proved by contradiction.

Ex: If $x \to y$ & $wy \to z$ then prove $xw \to z$

Proof: Assume $xw \not\to z$

then

w	x	y	z
w_1	x_1	y_1	z_1
w_1	x_1	y_1	z_2

then $wy \not\to z$

So, by contradiction $xw \to z$.

Imp.: Precise

Closure: Given a set of FD, F and a set of attributes X, the closure of X w.r.t F, written as X_F^+, is the set of all the attributes whose values are determined by value of X because of F.

$X_F^+ = \{w \mid X \to w\}$

↳ closure ↔ set of attr. X determines.

Here also abuse of notation $\{A\}^+ = \{A,B,C\}$ is written as $A^+ = ABC$ for conv.

PAGE NO. :
DATE : / /

Ex: FD set F = {a→b, b→c, d→b}

then

$b^+ = bc$, $c^+ = c$, $d^+ = dbc$, $(ad)^+ = abcd$, etc.

NOTE ⟷ If $(x^+) = R$ (i.e. all attributes) then x is surely superkey, not mandatorily CK.

So. K is SK if K determines all attributes in relation & it is CK if no proper subset of it is SK.

Que: F = {A→B, B→C}, R(A, B, C, D) then CK of R?

→ As. A & D can't be determined from any FD then they will be part of CK.

$(AD)^+ = (ADBC)$, so AD is CK.

NOTE ⟷ Any attribute y which is not present in RHS of any FD can be determined by itself only so each CK will have y.

→ Steps to know key from FD set: if A is

- Find A^+
- A^+ must contain all attributes of relation
- make sure no subset of A determines all attributes.

→ If $(xy)^+ = (a, b, c, d)$

then xy→a, xy→bc, xy→cd, xy→c, etc.

PAGE NO. :
DATE : / /

★ **Closure of Functional Dependency:** Formally, the set of all dep. that include f as well as all dependencies that can be inferred from f is called the closure of FD set f.

F^+ = set of all FDs inferred from f.

Ex: R(A, B, C, D) and $f = \{A \to B, B \to C\}$ then F^+ is?

→ $F^+ = \{A \to A, B \to B, \ldots, A \to B, B \to C, A \to C, AB \to A, \ldots\}$

So many FD's. Trivial too.

If $A^+ = ABC$ then FDs such that A in LHS are $2^3 - 1 = 7$, i.e. all non-empty subsets of ABC.

★ **FD Cover:** We say that a FD set F covers another FD set G if every FD of G can be inferred from F.

F covers G if $G^+ \subseteq F^+$.

→ F is minimal cover of G if F is smallest FD set (in count) that covers G. Minimal cover may not be unique. Many ways to find minimal cover and diff way give diff minimal cover.

→ FD set F & G are eq. if F covers G and G covers F.

i.e $F = G \to F^+ \supseteq G^+$ and $G^+ \supseteq F^+$

Ex:1 $F: \{a \to b, b \to c\}$, $G: \{a \to b, a \to c\}$, R(a, b, c, d)

F covers G as $a^+ = bc$ so $a \to b$ & $a \to c$

G don't cover F as $b^+ \neq c$ so $b \not\to c$.

So F & G are not equivalent.

PAGE NO. :
DATE : / /

Ex:2 F: {A→B, B→ACD, CD→B}

G: {A→B, B→CD, CD→A} R(A,B,C,D)

F covers G as from F:

$A^+ = B$, $B^+ = CD$, and $CD^+ = A$.

G covers F as from G:

$A^+ = B$, $B^+ = ACD$, $CD^+ = B$.

So F & G are equivalent.

★★ Normalisation:

→ Prime attribute (key attribute): An attribute is prime attribute if it is part of atleast one CK.

→ Non-Prime attribute: Not part of any of the CK.

Ex: F: {A→B, B→C}, R(A,B,C)

→ CK = $A^+ = ABC$.

So prime att. {A} & non-prime {B, C}.

★ Imp: Types of FDs:

1. Trivial & Non-Trivial FDs: Trivial means obvious things. So trivial FDs always holds or never holds.

$\alpha \to \beta$ is trivial FD if $\alpha \supseteq \beta$

PAGE NO. :
DATE : / /

Trivial FDs are not much intresting. as they are obvious.

Ex: $AB \to BC$

$AB \to B$ & $AB \to C$

$AB \to B$ → Trivial

$AB \to C$ → Non-Trivial

NOTE: If $Z = LHS \cap RHS$ then Z can be removed from RHS. as it is trivial part.

i.e. $\alpha Z \to \beta Z \longleftrightarrow \alpha Z \to \beta$

→ $\alpha \to \beta$ is non-trivial FD iff $\alpha \cap \beta = \phi$

NOTE: Now in further concepts we'll consider non-trivial FDs only with RHS being a single attribute.

II Partial FD & full FD: $\alpha \to \beta$ is full FD i.e. β is fully dependent on α iff any proper subset of α can not determine β.

$\alpha \to \beta$ is full FD iff $\forall A \in \alpha \quad (\alpha - A) \not\to \beta$

→ If FD is not full FD, then it is partial FD i.e. $\alpha \to \beta$ is partial iff $\exists A \in \alpha \quad (\alpha - A) \to \beta$.

Ex: $F: \{CD \to E, D \to A, A \to E\}$ In F $CD \to E$ is partial FD as $D \to E$ and $D \subseteq CD$.

→ FD with one attribute only, on LHS is fully FD because it has no proper (non-empty) subset.

$A \leftrightarrow B$ means $A \to B$ & $B \to A$

$\alpha_1 B_1$ $X Y$ PAGE NO. y set of attr.
DATE → our conv

→ Basically, in fully non-trivial FD $\alpha \to B$ if we remove anything from α then dependency does not anymore hold.

III Transitive FD: Transitivity in general

$x \to y \to z$ (called transitive edge.)

* → $X \to A$ is transitive FD if $\exists y$ such that $X \to y \to A$ and $A \notin X$, $A \notin Y$ where A is a single attribute

Que: If $x \to y \to A$ then can $A \to x$?

⇒ No,

Proof: Assume $A \to x$

then $y \to A$ & $A \to x$ by transitivity $y \to x$, and that's a contradiction. So $A \not\to x$

Hence Proved.

Ex: $F: \{A \to C\}$

a) is $AB \to C$ a transitive FD?

⇒ Yes:

$AB \to A \to C$

So, transitive FD.

Also partial FD as subset can determine.

b) is $A \to C$ also transitive FD?

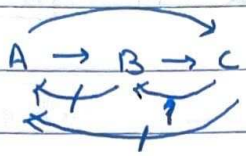
⇒ No.

$A \to ? \to C$ (can't be C because $A \to C \to C$ but $C \in C$ Not allowed.)

Not transitive

PAGE NO. :
DATE : / /

In transitive FD



$B \not\to A$, so $C \not\to A$ and $C \to B$ is neither prob. nor req.

Que: TIF?

a) Every partial FD is transitive FD?

⇒ No

Ex: $\{AB \to C ; A \to C ; A \to B\}$
↳ Partial. but not transitive

as $AB \to A \to C$ ... $AB \to B \not\to C$
↳ Due to this non-transitive.

b) Every transitive FD is partial FD?

⇒ No

Ex: $\{A \to B ; B \to C ; A \to C\}$
↳ Transitive & full FD

NOTE: With relation of 2 attributes we can not have partial or transitive (non-trivial FDs), we need at least 3 attributes.

✓

Atomic ↔ indivisible

PAGE NO :
DATE : / /

★ **Normal Forms:** We'll see why? later. Just enjoy it today.
A relation can be in 1NF, 2NF, 3NF or BCNF
↳ for Gate.

I **1st Normal form (1NF):** Relation R is in 1NF iff the domains of all attributes of R are atomic.

It's already present in schema. So no worries of it.

1NF is now considered part of formal definition of relation in relational model.

Every relation is trivially in 1NF. It's only NF defined w.r.t domain, rest are defined w.r.t non-trivial FDs.

II **2nd Normal form (2NF)** It do not allow a special type of partial FD:

CK → Non-prime attribute. → Partially.
↳ to be in 2NF, it must be absent.

To be in 2NF, you must be in 1NF.

Other partial FDs and all Full FDs are allowed.

Qu: Are partial FD not ~~att~~ all allowed in 2NF?

⇒ No, specific partial FD CK → non prime attr. are not allowed.

Qu: Is FD CK → non prime attr. not allowed in 2NF?

⇒ Not allowed only if it is partial i.e. subset of CK also determine non-prime attr.

PAGE NO.:
DATE: / /

* Def:

A relation is in 2NF if every non-prime attribute is fully dependent on every candidate key. (Think) ⇒ Analyse

Ex: F: {A → B, BC → E, ED → A} , R(A, B, C, D, E).

a) List all keys of R.

⇒ CD not in RHS so part of every CK.

$CD^+ = CD$ not ck.

$(ACD)^+ = (ECD)^+ = (BCD)^+ = ABCDE$,

So C.keys of R are ACD, BCD & ECD

b) Is it in 2NF?

⇒ As no non-prime attribute CK → non prime FD do not exist so it is in 2NF.

* NOTE ⟷ If all CKs are of single attribute or there is not any non-prime attribute then, Relation is in 2NF. for sure

Ex: Check 2NF: A) FD: {AB → CDE, C → D} R(A, B, C, D, E)

⇒ CK = AB

Non prime attr. CDE

A ↛ C, A ↛ D, A ↛ E, B ↛ C, B ↛ D, B ↛ E

So it is in 2NF.

B) FD: {AB → CD, B → C} R(A, B, C, D)

⇒ CK = AB

B → C So AB → C is partial FD. C and C is non-prime So not in 2NF.

Remember.

PAGE NO :
DATE : / /

Subset of CK → Non-prime
↳ Violates 2NF.

→ Violation of 2NF occurs when
Some CK → some non-prime attribute.
↳ This FD is partial then 2NF violated

OR.

* Proper subset of some CK → Some Non-prime Attribute.
↳ This FD violates 2NF.

→ To check if relation is in 2NF:

I. Find all CKs & Prime & non prime attributes
II Check if ~~prime att.~~ Some proper subset of CK can determine some non-prime attribute.

NOTE ⟷ We can have partial FDs of following type in 2NF:
CK → prime attr.

Ex: F = {AB ↔ CD ; B → D} R(A,B,C,D)

Here CK: AB, CD
no Non-prime attr. so in 2NF.
& AB → D is partial FD as B → D but D is prime attribute. ✓

So, it's possible for a prime attr. to be partially dependent on CK (of which it is not a component, to be non-trivial).

PAGE NO. :
DATE : / /

Que: Can a non-prime attribute determine prime attribute?

⇒ No never,

Proof: Assume $A \to B$ and A is non-prime & B is prime

it means XB is a CK & A is not part of any CK

$A \to B$

then $XA \to XB$

XB is CK so XA is also a CK, so A is part of CK

Contradiction. So $A \not\to B$

Hence proved.

Que: Can a non prime attr. determine another non prime attribute?

⇒ Yes, No problem.

III **3rd Normal form (3NF):** A relation R is in 3NF if it is in 2NF and if no non-prime attribute is transitively dependent on any CK.

So every non-prime attribute is transitively dependent on every CK.

→ Violation of 3NF.

Some CK —Transitively→ Some non prime attribute.

⇕ Eq. to (Proved later)

Not super key ⟶ Some non-prime attributes

↳ Such FD must not exist to be in 3NF.

PAGE NO. :
DATE : / /

Remember

Not a SK → Non prime attribute violates 3NF.

Proof: let α be not a super key (so can't be CK also, so $\alpha \neq$ CK) & A be non-prime attribute.

If $\alpha \to A$ then.

CK → α → A

If α was SK then α = CK, so need to be non SK.

So CK $\xrightarrow{\text{trans.}}$ A. is eq. to $\alpha \to A$. (Not SK.)

Now non-super keys are:

a) Non prime attributes sets
b) Prime + Non prime
c) Subset of CKs.

So.

→ Non Prime X → Non prime Y is problem in 3NF as CK → non prime X → Y

→ Non prime → prime is impossible.

→ Prime + Non prime → Non prime is problem in 3NF (Not SK)

⇒ So to check 3NF just check:

Not SK → Non prime attr.

↳ This FD must not exist.

→ If attr. is transitively dep. on CK then it is det. by a non-SK also. (proved above this)

PAGE NO.:
DATE: / /

So, 3NF violations are:

a) non prime → non prime

b) prime + non-prime → non prime
↳ NKA + SK

Can't be SK

c) Subset of CK → non prime.
↳ Also of 2NF.

So 3NF already covers 2NF i.e. if Relation in 3NF it is already in 2NF.

Que: Check 3NF. (assume attr. from FD set).

a) $A \to C$; $B \to A$; $A \to D$; $AD \to C$

⇒ CK = B, in 2NF bcz no subset of B possible.
but $A \to C$ violates 3NF. (NP → NP) so not in 3NF.

b) $C \to D$; $CD \to A$; $AB \to C$; $BD \to A$

⇒ CK = AB ; BD ; BC no non prime attr. so violation not possible in 3NF.

NOTE If no non-prime attribute then it is in 3NF for sure.

Que: Can we have transitive dep. in 2NF?

→ Yes. Of type CK Transitive Prime attributes.

Ex: $AB \leftrightarrow CD$, $A \to C$. } Non-prime attr. do not exist so in 3NF
$AB \to C$ is transitive FD as: $AB \to A \to C$

NPA
↓↓
Non Prime Attribute

PAGE NO. :
DATE : / /

In 3NF ⟷ In 2NF

2NF violated ⟷ 3NF violated

Ex: Check 3NF:

a) $B \to C$; $AC \to D$; $ABD \to C$; $BCD \to A$.

⇒ CK: BD, BA,

Non prime attr. is C

and $B \to C$ so not in 2NF & not in 3NF

(B: Not SK; C: NP Attr.)

$BD \to B \to C$ so CK ⟶ NP attr. so not in 3NF.

b) $AB \to C$; $ABD \to C$; $ABC \to D$, $AC \to D$

⇒ CK = AB

NP Attr.: CD.

$A^+ = A$, $B^+ = B$ } so no partial FD of type CK → NPA, so in 2NF

$AC \to D$ } so not in 3NF

(A: Not SK; D: NPA)

↳ Comb^n of prime + NP attr.

Q: If NPA is non-trans. dependent on CK then NPA is also fully dependent on CK?

⇒ Yes.

Proof: Let NPA partially dep. on CK. & α be subset of CK

$\alpha \to$ NPA.

$CK \to \alpha \to$ NPA so transitive dep.

So Contradiction.

PAGE NO. :
DATE : / /

IV Boyce Codd Normal form (BCNF): A relation R is in BCNF iff all non-trivial FD's are of form:

Superkey → y.

i.e. non-superkey must not determine anything.

→ Violation of ~~2nd~~ BCNF:

Not SK → Prime / NPAttribute.

↳ such FD must not be there.

→ In 3NF this FD was allowed if RHS was Prime attr. but not allowed in BCNF.

→ So BCNF covers 3NF.

In BCNF → In 3NF

BCNF Violate ↛ 3NF violated

→ A relation is in 3NF & not in BCNF iff LHS of FD is not SK & RHS is prime attr.

i.e. Not SK → Prime Attr.

↳ Allowed in 3NF but not in BCNF.

→ Any relation with two attribute is in BCNF surely.

Proof: R(A,B), then non-trivial FD's possible:

Case I: No FD, then in BCNF. , CK = AB

Case II: {A → B, B → A}, CK = A, B, so no non SK, so in BCNF.

Case III: {A → B} here CK = A, non SK possible B ↛ anything so in BCNF.

PAGE NO. :
DATE : / /

☆ **Natural Join:** Natural way of joining two relations / tables

Just match the common attributes and join them.

Ex: S =

Name	SSN	Grade
A	121	2
A	123	3
B	101	4
C	106	2
* D	107	1

T =

Grade	Salary
2	80
3	70
4	70
* 5	60

then S & T's natural join would be: $S \bowtie T$ (→ Natural Join)

Name	SSN	Grade	Salary
~~A~~	121	2	~~80~~
A	123	3	~~70~~
B	101	4	70
C	106	2	80

In jointed table we write common attr. once only & more than 1 common attr. can also be there. And non-matching rows are dropped in join.

If no common attr. then we do Cross product i.e. combine every row of S with every row of T.

PAGE NO. :
DATE : / /

Redundancy: A database is said to be poor if it is redundant.

FD's may create redundancies:

Ex: City → State.

x	City	State
1	Jpr	raj
2	Kota	raj
3	Jpr	Raj
4	Jpr	Raj
5	Jpr	Raj

Redundancy due to FD.

Redundancy do not only waste space. It may lead to anamolies (problems) also:

a). Insertion Anamoly i.e. If x → y and already some tuple has x kk and respective y pp then new kk at x should have only pp at y.

b). Deletion anamoly i.e. if we delete last tuple repr. attr. value then that value of attr. is lost.

c). Update anamolies i.e. if we forget to update some attr. and update other attr. may cause inconsistency.

d) Redundant Storage.

The solution for this is DECOMPOSITION.
i.e. decompose table in various tables.

PAGE NO :
DATE : / /

Decomposition: Essential idea of decomp. is that replace a relation with collection of smaller relations to address problems arising from redundancy.

Two properties of decomposition are of particular intrest:

a) **Lossless - Join Property**: It enables us to recover any instance of the decomposed relation from corresponding instances of the smaller relation using join w/o any extra (spurious) tuples.

b) **FD preserving property**: It enables us to enforce any constraint on the original rltn by simply enforcing constraint on each of the smaller relation. i.e. Decomp. is FD preserving if FDs of bigger relations holds in smaller also.

NOTE: Decomposition is not even valid if all attributes are not preserved

i.e. Schema union $(R_i) = R$
↳ Mandator for decomposition.

→ Every attr. of R should be present in atleast one of the new decomposed relations.

→ If R is decomposed in two tables only then it's a binary decomposition.

PAGE NO. :
DATE : / /

→ Spurious tuples are those tuples which were not originally in relation but in join of decomp. relation. This is loss of information. It is present in lossy decomposition.

→ Lossless decomp. do not give spurious tuples nor eat our original tuples.

Ex:

a	b	c
1	2	3
2	2	3
3	2	2

Here
$[R_1(a,b) \bowtie R_2(b,c)]$ is lossy decomp. (check)
$[R_1(a,b) \bowtie R_2(a,c)]$ is lossless decomp.

R_1

a	b
1	2
2	2
3	2

R_2

b	c
2	3
2	2

$R_1 \bowtie R_2$

a	b	c
1	2	3
1	2	2
2	2	3
2	2	2
3	2	3
3	2	2

Spurious tuples

Lossy

R_1

a	b
1	2
2	2
3	2

R_2

a	c
1	3
2	3
3	2

$R_1 \bowtie R_2$

a	b	c
1	2	3
2	2	3
3	2	2

Lossless decomposition.

→ In loss decomp. we get extra tuples but our original tuples never lost.

So, $R_1 \bowtie R_2 \bowtie \dots R_n \supseteq R$ ∀ Any decomp.
= → lossless
→ lossy

PAGE NO :
DATE : / /

→ FDs, NFs, Keys, Decomp, etc. are properties of schema & not instance.

⇒ Lossless Decomposition:

→ In a binary decomposition, we check if it is lossy or lossless (cannot confirm by seeing instance, need to check schema) by following property:

* Binary decomp. is lossless iff set of common attr. of R_1, R_2 is a superkey in atleast one of R_1 or R_2

i.e. $R_1 \cap R_2 \rightarrow R_1$ OR

$R_1 \cap R_2 \rightarrow R_2$.

↳ To be lossless.

↳ Only for binary decomposition.

NOTE ⟷ FDs are property of whole DB not of any particular relation.

Ex: $F: \{A \rightarrow BC, CD \rightarrow E, B \rightarrow D, E \rightarrow A\}$ $R = \{A, B, C, D, E\}$

Check following decomp is lossless or lossy?

a) $R_1 = \{A, B, C\}$, $R_2 = \{A, D, E\}$

⇒ Common Attr. is A $A \rightarrow BC$ so A is key of R_1 so lossless

b) $R_1 = \{A, B, C\}$ $R_2 = \{B, C, D, E\}$

⇒ common attr is BC $(BC)^+ = BCDE$ so $BC \rightarrow R_2$ so lossless

c) $R_1 = \{A, B, C\}$ $R_2 = \{C, D, E\}$

⇒ Common attr. is C $C^+ = C$ so $C \not\rightarrow R_1$ & $C \not\rightarrow R_2$ so lossy

PAGE NO. :
DATE : / /

→ By looking at instance, we can't say lossless bcz, it may get lossy after sometime. We check FD. FD is for whole DB not relation. So, Check lossy or lossless on schema, not on instance.

⇒ **Dependencies Preservation:**

→ If FDs of DB is F & of R_1 is F_1 & R_2 is F_2 then $(F_1 \cup F_2)^+ \supseteq F^+$ is never possible.

→ Decomp. of relation schema R with FDs is set of small relations R_i with FDs F_i. F_i is subset of F^+ that includes attributes of R_i.

Ex: $F = \{A \to BC, CD \to E, B \to D, E \to A\}$, $R = \{A, B, C, D, E\}$
$R_1 = \{A, B, C\}$ & $R_2 = \{A, D, E\}$
What is F_1 and F_2?

⇒ From F,

$A^+ = ABCDE$
$B^+ = BD$, $(BC)^+ = BCA$
$C^+ = C$
$D^+ = D$
$E^+ = A$

So $F_1 = \{A \to BC, BC \to A\}$ } Only non-trivial.
$F_2 = \{A \to DE, E \to AD\}$
↑ R_2 can preserve only these FDs.

Now if $(F_1 \cup F_2)^+ = F^+$ then decomp. is FD preserving else it is not so. $F_1 \cup F_2$ should cover F for decomp. to be FD preserving.

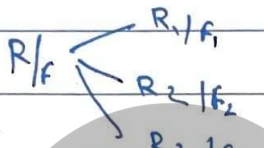
Here $CD \to E$ & $B \to D$ are not preserved in F so not FD preserving.

PAGE NO. :
DATE : / /

→ In actual we did not lost FD, it will be present in join obviously (as not lossy) but this FD can't be enforced w/o joining tables. so we say non-FD preserving decomposition

↗ No red. & no anomaly

→ Decomposed table have many benefits but also a loss which is we need to join table (which is costly) to get some information.

→ 

If $F_1 \cup F_2 \cup F_3$ covers F then decomp. is FD preserving. (obviously F covers them, there can't be extra FD in union)

→ Most imp. this is lossless, if it's lossy decomp. then FDs are actually lost after join also.

→ Dep preserving $\rightleftharpoons$ FDs enforced w/o joining tables

Que: $R = \{A, B, C, D, E\}$ $F = \{A \to BC, B \to D, E \to A, CD \to E\}$

$R_1 = \{A, B, E\}$, $R_2 = \{B, C, E\}$

Find F_1 & F_2?

⇒ $F_1 = \{E \to AB, A \to BE\}$

$F_2 = \{E \to CB, BC \to E\}$ ↓ Don't forget this

$A^+ = ABCDE$ ↗ lossless

$B^+ = BD$

$E^+ = ABCDE$

$C^+ = C$

$BC^+ = BCDE$ ✓

Phantom Tuples ⇌ Spurious Tuples

Lossless Join Decomp. ⇌ Non-Additive Join Decomp.

PAGE NO.:
DATE: / /

F^+	$\supseteq$	$[\text{Union}(F_i)]^+$	Always
F^+	$\subset$	$[\text{Union}(F_i)]^+$	Impossible
F^+	$=$	$[\text{Union}(F_i)]^+$	Dep. preserving
F^+	$\supset$	$[\text{Union}(F_i)]^+$	Dep Non preserving

Que: Check FD preserving or not

$F = \{A \to B, B \to C, A \to C\}$ $R_1\{A, B\}$ $R_2\{B, C\}$

$F_1 = \{A \to B\}$

$F_2 = \{B \to C\}$

$(F_1 \cup F_2)^+ = \{A \to B, B \to C, A \to C\}$ So $\equiv F$. So FD preserving.

→ Amount of decomposition to do is decided by how much normalisation you need. In which NF your DB must be in.

→ BCNF has 0% redundancy because of FD, but other redundancies (Multivar FD) are present still.

R — In 1NF

/ | \ } BCNF decomp

R_1 R_2 R_3

In BCNF

→ NF tells how good is your DB. If in BCNF it's best. So we decompose to reach particular NF.

→ NF of relation is highest NF i.e if in 3NF then said to be in 3NF (Not 2NF despite being in 2NF also).

PAGE NO. :
DATE : / /

★ Non-Binary Decomposition:

For this way to check FD preservation property, method is same. But to check lossless or lossy it is diff from binary decomposition. We do it by chase test.

(Works for any (binary / non-binary) decomposition.)

Que: If attr. are not preserved then we say decomposition is ___?

⇒ Invalid.

- If $R_1 \cap R_2 = \phi$ in binary decomp. then surely it will be a lossy decomposition as $\phi \not\to R_1$ and $\phi \not\to R_2$.

⇒ Succesive combining relations to check lossless decomp say method X:

→ It is one way i.e. if it says lossless then lossless but if it says lossy then too it can be lossless or lossy.

→ Say R is divided as R_1, R_2, R_3, R_4, then if there's a way to combine these subrelations by combining two subrelations at a time in lossless manner then decomp. is lossless, if there is no way then we can't say anything.

For Ex: $F: \{A \to B, B \to C, C \to D, D \to A\}$ decom. as $R_1 = AB$, $R_2 = BC$, $R_3 = CD$.

$R_1 \cap R_2 = B \Rightarrow B^+ = BC$ (key) so losslessly join R_1 R_2 as ABC

now $\underset{R_{12}}{ABC} \cap \underset{R_3}{CD} = C \Rightarrow C^+ = CD$ (key) so losslessly joins as ABCD

So lossless join. ✓

PAGE NO. :
DATE : / /

But this process can be lengthy and as we already know, one way as well.

for Ex: $F: \{A \to C, B \to C, C \to D, DE \to C, CE \to A\}$

$R_1 = AD$, $R_2 = AB$, $R_3 = BE$, $R_4 = CDE$, $R_5 = AE$

Here our method X will not give any method to join losslessly. Try any way. But still it is lossless.

↳ Verify using Chase algo.

→ Chase algorithm is used to check if given decomp. is lossless or lossy.

☆ Chase Algorithm

$(A_1 -- A_n)$

I/P: Schema R_{A_1}, FD set and decomposition $R_1 -- R_k$

O/P: Decomp. is lossless or lossy.

Method: Construct a table with n cols (as n attr. in R) and k rows (as k diff. decompositions). In i^{th} row & j^{th} column put symbol a_j if A_j is in R_i & not put symbol b_{ij} there.

Repeatedly consider each FD $X \to Y$ from F untill table can't be updated. For each fd, we look for rows that agree in all the columns for the attr. of X. If we find two such rows equate the symbols of those rows for attr. of Y.

If after modifying rows of table, we discover that some row has become $a_1 -- a_k$ then join is lossless else lossy.

PAGE NO. :
DATE : / /

Ex:1 $R_1 = ab$, $R_2 = ac$, $R_3 = bcd$, $R_4 = ade$

Initial setup:

	A	B	C	D	E
R_1	a	a			
R_2	a		a		
R_3		a	a	a	
R_4	a			a	a

Now F: { $A \to C$, $BC \to E$, $E \to D$ }

	A	B	C	D	E
R_1	a	a	a	a	b
R_2	a		a		
R_3		a	a	a	b
R_4	a		a	a	a

Due to row 1 it is lossy.

1) Initial setup, Rows: subrelations, Columns: Attributes.
2) Put 'a' in cells for subrelations containing attribute.
3) Fill table using FDs if $x \to y$ then if two tuple has same x value then same y value is filled.
4) Repeatedly apply step 3 untill no changes could be made
5) If row of a's occur then lossless else lossy (terminate when any 1 row is full).

Relation Instance
≈
Relation Extension

PAGE NO. :
DATE : / /

Ex:2 $F = \{A \to C,\ B \to C,\ C \to D,\ DE \to C,\ CE \to A\}$

$R_1 = AD$, $R_2 = AB$, $R_3 = BE$, $R_4 = CDE$, $R_5 = AE$

	A	B	C	D	E
R_1	a		b	a	
R_2	a	a	b	a	
R_3	a	a	a	a	a
R_4	a	a	a	a	a
R_5	a		a	a	a

$DE \to A$.

So lossless.

← Practice its PYQs :→

→ FD satisfied by instance
↓ ↑
FD satisfied by schema

→ If instance is given we can tell potential ck

Gate 2000

Qu: Which FD's are satisfied by below instance?

X	Y	Z
1	4	2
1	5	3
1	6	3
3	2	3

⇒ $YZ \to X$ and $Y \to Z$

PAGE NO :
DATE : / /

⇒ **Implied FD:** Given FD set F, does F imply FD $X \to Y$.

$\Downarrow$

$F \models (X \to Y)$

↳ inference symbol of prop. logic.

Que: $f: \{A \to B, B \to C\}$

a) $F \models (A \to C)$?

⇒ Yes,

From F, $A^+ = ABC$ so $A \to C$ ✓

So F infer $A \to C$. ✓

b) $F \models (AB \to C)$

⇒ Yes,

From F $(AB)^+ = ABC$ so $AB \to C$ ✓

c) $F \models (C \to A)$.

⇒ No,

From F $C^+ = C$ so $C \not\to A$ from F.

F don't infer $C \to A$.

Que: If $F: \{X \to Y, Z \to B\}$ & $Z \subseteq Y$, X Y, Z, B are set of attr. then $F \models X \to B$?

⇒ As $Z \subseteq Y$ trivially

$Y \to Z$.

So by transitivity in F $X \to B$ is True Inference.

Extraneous attribute: Attribute we can remove from FD set.

PAGE NO.:
DATE: / /

To check if FD set F implies fd $f: X \to Y$ i.e. $F = \{f\}$ check x^+ from F if Y is in x^+ then $F = \{f\}$ else $F \neq \{f\}$.

★ Cannonical / Minimal Cover of FDs: ⇌ Irreducible set of FDs.

→ Informally f_1 is said to be stronger than f_2 if f_1 implies f_2.

Ex: $(A \to C)$ is stronger than $(AB \to C)$ as

$(A \to C) \longrightarrow (AB \to C)$

→ Minimal cover of FD F is irreducible set of FDs that do not contain redundant FD and any FD can't be removed from the set.

Many minimal covers of FD set are possible & diff minimal covers can have diff no. of FDs also.

Minimal Cover and actual FD set are equal.

Let's do it.

⇒ Extraneous attribute on LHS of FD:

$$\left\{ \begin{matrix} A \to C \\ AB \to C \end{matrix} \right\} \equiv \{A \to C\}$$

↓

B is redundant.

In FD $XA \to Y$ A is extraneous if $X \to Y$. i.e. x^+ contains y.

We can remove this extraneous attr. from FD.

Minimal Cover is not unique (de

PAGE NO :
DATE : / /

Ayz

F = { AC → G ; ACD → B ; BC → D ; CG → B ; CG → A ; CE → A ; CE → G ; D → E ; D → G }

Find to remove extraneous attr. on LHS of each FD of F.

⇒ In AC → G, A^+ don't contain G so C is req. & C^+ don't contain G so A is req.

ACD → B, $(AC)^+$ = GACB, B is extraneous

Now $A^+ = A$ & $C^+ = C$ so AC is req.

$D^+ = DEG$

So AC → B. ✓

and so on.

⇒ Removing redundant FDs:

① make sure that every FD contains single attr. on RHS to make things easy.

② In FD set F, FD X → A is redundant if

F − (X → A) implies X → A

i.e. F and X → A and F with X → A is equivalent then X → A is redundant.

Check this for each FD that from

[F − (X → A)] → (X → A)

(implies)

= }
X → A
= }

If these can imply X → A then X → A is redundant.

PAGE NO. :
DATE : / /

So $(f - (X \to A))^+ \equiv f^+$ then $X \to A$ is redundant.

Ex In que xyz on prev page
$AC \to G$ is not redundant as
from $f - (AC \to G)$ gives $(AC)^+ = AC$ So $AC \to G$ is req.
but,
$CD \to B$ is redundant FD as
from $f - (CD \to B)$ we get $CD^+ = CDB$ So $CD \to B$ means $CD \to B$ is redundant.

NOTE ↔ If in FD set only a single FD determines an attr. then it is not redundant.
for ex: $CE \to A$ in prev. ex is not redundant as A is given only by CE.

⇒ Algorithm for minimal Cover: (No rules of order)

Step-0 Remove trivial FDs

Step-1. Split the right part such that RHS has only one attr. in each FD

Step-2 Remove extraneous attribute from LHS of FDs

Step-3 Remove redundant FDs

→ There are many ways to find minimal cover, ours is least error prone.

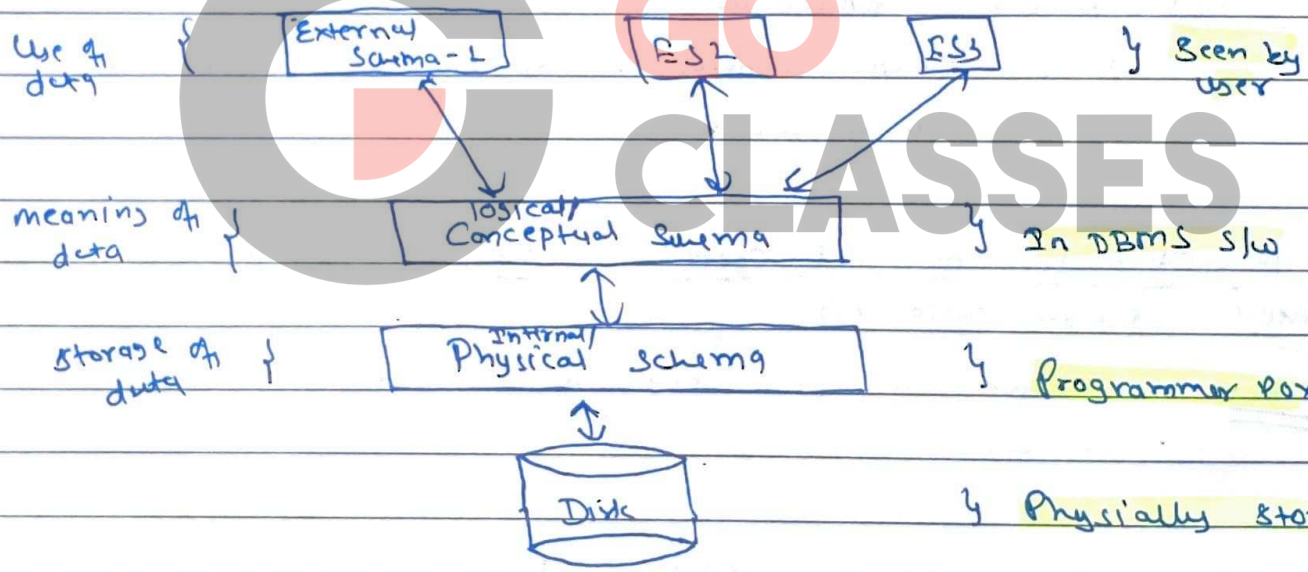
→ Asking no. of FDs in min. Cover is nonsense as minimal cover is not unique.

PAGE NO. :
DATE : / /

★★ Three levels of Schema

→ In a db, whole DB must not be visible to all types of users. There should be a hierarchy. So diff type of users have diff views.

→ The DB description consists of a schema at each of three levels of abstraction: Conceptual, Physical & External schemas.



Levels of Abstraction of DBMS

★ **Conceptual / Logical schema:** What type of data stored in DB, what relations, tables stored in DB. This schema describes the stored data in terms of data model of DBMS.

PAGE NO.:
DATE: / /

→ Most of the application programmer are written on the basis of conceptual schema.

→ Like Student (S-id, int; name, string), now in DAPs query will use this info, it do not care stored as B tree/ or B+ tree or in which order, etc.

B Physical Schema (Internal schema): It specifies storage details. It deals with actual storage in disk. It decides to store sorted on what field, decides index, etc. It deals with which DS to use like B tree or B+ tree.

C External schema (Viewed schema): This is what user sees. It retains data privacy, May convert DoB → Age. Show joined tables etc.
(DoB: Actually stored; Age: calculated from CS)

It is dependent on user requirements.

→ Es derived from Conceptual schema (CS). It is not actually stored physically. It's a virtual table.

→ CS is rarely change (schema remains same) and is based on physical schema.

PAGE NO. :
DATE : / /

Que: Why three levels of schema?

Ans. 1. Abstraction
2. Privacy.
3. Data Independence.

★★ Data Independence: ^ Programs should be indep. from details of storage & data representation.

→ Physical Data Independence: Independence of cs from physical changes. i.e. Change physical schema w/o worrying about conceptual schema.
It is basically a measure of how much physical schema we can change w/o affecting application programs. It is not hard to achieve.

→ Logical Data independence: Changes in cs should least affect the external schema.
It is basically a measure of how much the conceptual schema we can change w/o affecting application program for ES. It is harder to achieve.

→ It is responsibility of DBMS s/w to provide max PDI & max LDI.

→ LDI is harder to achieve than PDI. bcz. changes in CS will surely effect app. program i.e. external schema.

PAGE NO. :
DATE : / /

Imp. Most common mistake while checking 2NF is not checking FDs in FD closure and only from given FD set.
So for every proper subset of CK, find its closure and check it must not contain non-prime attribute.

Gate CS 20

Que: Tell NF of table R?

a) R has a non-trivial FD $X \to A$ where X is not SK & A is prime attribute?

→ It is in 3NF & not in BCNF.

b) R has a non-trivial FD $X \to A$ where X is not SK & A not proper subset of any key & A is non prime attribute.

→ It can be in 2NF & not in 3NF.

c) R has a non-trivial FD $X \to A$ where X is proper subset of key (not SK) and A is NPA.

→ It is in 1NF & not in 2NF.

d) A cell in R holds a set instead of atomic value?

→ It is not in 1NF.

→ for 3NF but not in BCNF we must have FD of type:

Not SK → Prime attribute

a. Non prime attribute (Not possible to have NPA → PA)

Any of these can lead to 3NF & not BCNF

b. Proper subset of CK ✓ → PA

c. Prime + NPA ✓ → PA

PAGE NO. :
DATE : / /

Qu: Comment on normal form: R(A, B, C, D, E)

F : { AB → C ; AD → BCE ; CD → ABE }

⇒ CK = AD & CD.

NPA = B, E

AB → C
↓ ↓
Not SK Primattr } So not in BCNF.

Non SK → NPA do not exist. So in 3NF.

So it's in 3NF & not in BCNF.

GateBycs.

Qu: Using armstrong's 3 Axioms prove:

CS DD a) {x → y, x → z} |= x → yz

→ **Proof:** Using armstrong's axioms (boring) (on GO)

Proof: Using closure ** My fav.

x^+ = xyz. (from FD set only)

So x → yz ✓

Proof: Make tables. (Sample)

CS DD b) {x → y, wy → z} |= xw → z

$(xw)^+$ = xwyz.

So xw → z ✓

PAGE NO.:
DATE: / /

(c) dd

c) $\{x \to y, zcy\} \models x \to z$

$\Rightarrow$ as. $y \supset z$

$y \to z$

So by trans. $x \to z$. ✓

★★ Understanding Redundancies:

→ Trivial FDs never create redundancies in the database.

→ Non-Trivial FDs may create " " " "

Ex: City → state in some table. Here repeting a city (which is info) repeats state also which is redundancy.

→ If city was superkey then it would not repeat at all and redundancy would not exist.

→ Soln is decompose table to make city, state a diff table making city as key. use city as FK in your initial table.

Ex: $\{AB \to C, C \to B\}$, only $C \to B$ cause redundancy as in $AB \to C$, AB is superkey already.

→ So not all non-trivial FDs create redundancy as FD with SK on LHS do not cause any redundancy.

→ So FDs of following type only creates redundancy:

$X \longrightarrow \alpha$

where it is not-trivial & X is not SK.

PAGE NO :
DATE : / /

→ Now coming to BCNF, every non-trivial FD has LHS as SK, so BCNF is free from redundancy caused by FDs. (other redundancy can be there, but out of our syllabus).

→ Take an example:

Courses (sid, cid) (many to many)

Movies (sid, movie) (many to many)

There will not be any redundancy here.

There is no non-trivial FD.

Now if we join the table:

CM (sid, cid, movie)

↳ There will surely be redundancy, but there is no FD here? So where did redundancy came from?

Analogy → Govindagwith 2 wives

** "when we put more than one many-to-many relationship in single table, there can be redundancy". This redundancy is known as multivariable dependency (MVD).

Redundancy

Due to FDs
↓
0% in BCNF

Due to MVDs
↓
0% in 4NF
↳ Not in GATE.

PAGE NO. :
DATE : / /

** X-NF Decomposition:

→ A relation with two att. will be in BCNF for sure (take any FD possible).

→ A set of relations $S = \{R_1, R_2, R_3\}$ is said to be in X-NF if each relation R_i is in X-NF.

→ If Relation is not in X-NF (say BCNF) then we make it into X-NF by decomposing it losslessly in X-NF sub relations

→ It is good to note that X-NF alone is not the goal, but the decomposition also must be lossless.

→ Decomp. is by default lossless to us as loss decomp is of no use.

So, if we are asked to:

a. Do X-NF decomposition then:
   Subrelations in XNF → Must
   Lossless Decomp. → Must
   Dependency Preserving → If possible.

b. Check if X-NF decomp. then:
   Check each subrelation is in X-NF.

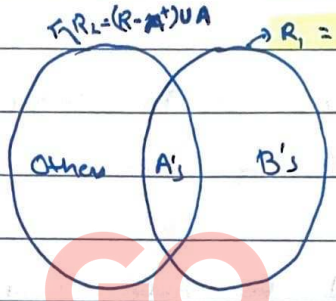
→ If we do not care about lossless then just make tables of 2 att. each for BCNF decomp. But we need lossless.

PAGE NO :
DATE : / /

BCNF Decomposition Algorithm:

It guarantees lossless decomp. into BCNF subrelations.

In this we look non-trivial FD $A_1A_2 \dots A_n \to B_1B_2 \dots B_n$ that violates BCNF i.e. $A_1A_2 \dots A_n$ is not a super key. We make a diff. table for it and ~~copy~~ move attr. func. determined by $A_1A_2 \dots A_n$, And in original table we have A's be leftover.



$R_2 = (R - A^+) \cup A$

$R_1 = A^+$

It is guaranting lossless as $R_1 \cap R_2$ is A and A is SK of R_1 bcz it has only its determined attributes.

Ex: 1 R(ABC) with A → B.

⇒ A → B is violating BCNF as A is not SK.

$R_1 = AB$
$R_2 = AC$ } lossless & in BCNF. decomposition.

Ex: 2 R(ABCD) with A → B, B → C.

⇒ CK is AD, so both are violating

Violation - A → B

$R_1 = AB$ — Projected FDs $f_1 = A \to B$

$R_2 = ACD$ — $f_2 = A \to C$ violating again

$R_{21} = AC$, $f_{21} = A \to C$

$R_{22} = AD$

So, AB, AC, AD is BCNF decomp.

PAGE NO. :
DATE : / /

Ex:2 R(ABCD) F = { A→B, B→C }

⇒ CK = AD so both are violating BCNF. So take any violation.

A→B

ABC AD

A→B, B→C ⟶ CK is A so B→C is violating again.

AB (A→B) BC (B→C)

So, AB, BC, AD is BCNF decomp

NOTE We may get diff. BCNF decompositions depending on the order of violation we choose.

Ex:3 R(A,B,C,D,E) F = { A→BC, CD→E, B→D, E→A }

⇒ CK = A, E, CD, BC

Violation here is only B→D

so

ABCDE

B^+ = BD
{B→D}
↳ B is SK

ABCE
{A→BCE, E→ABC, BC→AE}
↳ All LHS SK

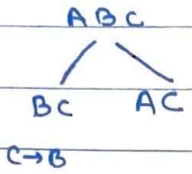
→ Imp. to learn projecting FDs

so in BCNF now BD & ABCE.

NOTE Lossless BCNF decomp. is always possible, but lossless with FD preservation is not always possible.

PAGE NO. :
DATE : / /

Ex:4 $AB \to C$; $C \to B$ here CK = AB, AC so $C \to B$ is violation



$AB \to C$ can't be preserved here.

BCNF + Lossless + FD preserving is impossible for this.

↳ This

↳ or this

(at 3)

→ for some relation we can get it, but not always possible

★ 3NF decomposition Algorithm : Also called the synthesis algorithm because it synthesized the 3NF sub relation instead of decomposing original relation

Features

→ This algo guarantees 3NF lossless FD preserving decomp we can achieve both lossless & FDP 3NF decomp. using this algo.

Steps:

① Find minimal cover of FDs say Fc.

② In Fc if LHS is same then merge the FDs like $X \to A_1$; $X \to A_2$ to $X \to A_1A_2$.

③ For each FD $X \to A_1A_2$ create sub relations XA_1A_2.

④ If none of the relations form prev. step contains a CK then take any CK and make one separate relation for it.

⑤ If $R_x \supset R_y$ then delete R_y

PAGE NO. :
DATE : / /

Ex:1. Consider relation R(A, B, C, D, E) with FDs {AB→C, C→B & A→D}
Do 3NF decomp?

CK = ACE, ABE

⇒ R is not in 3NF as in C→B, C is not SK so B is NPA.

① Minimal Cover. → Already minimal cover.

② SR_1 = ABC, BC, AD

③ Not any CK so, SR_1 = {ABC, BC, AD, ACE}

④ ABC ⊇ BC so

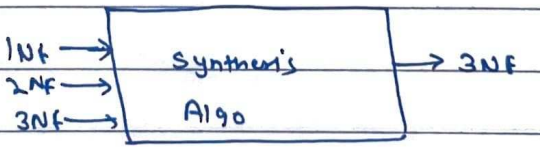
finally Subrelations are ABC, AD & ACE

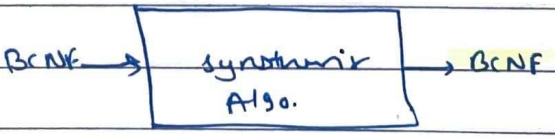
NOTE: Checking if in 3NF takes more time than actually synthesising 3NF relations. So we synthesise w/o checking in real life. It can be directly applied w/o checking if already in 3NF or not.

Qu: If R is in BCNF & we apply synthesis algo on it then new R is non-BCNF possible?

⇒ Not possible (think)

So,





→ 3NF + Lossless + FD preserving ✓ Always Possible } we have algo
BCNF + Lossless ✓ Always Possible
BCNF + Lossless + FD preserving ✗ Sometimes possible ✓
Lossless + FD preserving ✓ Always Possible.

PAGE NO. :
DATE : / /

imp.
* → If our decomp. is FD preserving then we can enforce FDs on decomposed subrelations directly, but if decomposition is not FD preserving then before insertion/updation (as deletion can't cause FD violation), we may have to join two or more tables to enforce some FDs, we can't just directly insert/update.

→ 3NF is preferred normal form as it reduces most redundancy and also guarantees FD preservation which BCNF do not & is req..

Module-2 Queries

PAGE NO.:
DATE: / /

☆ Relational Algebra

→ Query languages are specialised languages for asking question or queries that involve the data in a DB.

→ We have two formal query languages associated with the relational model:

a) Relational Algebra
b) Relational Calculus
   ↳ TRC
   ↳ DRC

} Theortical i.e. cannot run & check them.

→ There are many practical/commercial QLs like SQL*, QUEL, Datalog etc. based on Theortical QLs made by DBMS providers.

☆ Procedural v/s Declarative Query language:

Analogy to algorithm & function respectively.

→ In dec. QL we just ask the question we do not tell how will you get it. It's DBMS resp. to fetch efficiently

Ex: SQL and Relational Calculus

→ In procedural languages we tell the procedure to fetch the data. (How + what)

Ex: Relational Algebra. (Here we tell first join then select or select then join, etc).

→ From user POV dec. language is easy & from compiler (for QL) POV. procedural language is easy.

→ SQL & RC are dec. bcz we tell what we want w/o telling how to do it. ✓

PAGE NO. :
DATE : / /

Relational Algebra: Queries in RA are composed using collection of operators, and each query describes a step-by-step procedure for computing the desired answer. i.e. queries in RA are specified in a procedural manner.

In relation algebra:

Variables ≡ Relation

Value ≡ Relation Instance.

Operators ≡ RA operators (Basic → Derived)

→ I/p and O/p of RA is query in relation instance. It often involve the comp. of intermediate results which are also relation instance.

Operators in RA: There are 6 basic operators in RA:

A. **Select Operator (σ):** Select desired tuples (filter rows)

Ex: $\sigma_{A>0.1}(User)$ — Condition; Relation

→ It does Horizontal partition.
→ Schema is uncharged i.e. degree/Order is same.
→ Cardinality (#Rows) changes.
→ Apply condition on each tuple individually, independently.

$\sigma_p(R) = \{t \mid t \in R \text{ and } p(t)\}$

where p is prop. calculus

PAGE NO. :
DATE : / /

Qus: Get student id with max. marks.

⇒ $\sigma_{marks > max marks}(student)$ ✗

↳ This is wrong as we can see only one row at a time.

NOTE ⟷ for any retrieval query original database never changes; new temp. relations may be created.

B. Projection Operation: Filters columns / attributes.

$\Pi_{(attr.list)}(Relation)$

↓ comma septd.

→ It's a vertical slicing.

→ #cols and #rows may change

↳ bcz it's cols rows duplicates

(#rows)

Qus: To make sure that cardinality of projected relation is necessarily same i/p relation, what can you say about the attribute list of projection operation?

⇒ Projected attributes must form a superkey for this.

* Properties of selection & projection:

a. Selection is commutative:

$\sigma_{p_1}(\sigma_{p_2}(R)) = \sigma_{p_2}(\sigma_{p_1}(R)) = \sigma_{p_1 \wedge p_2}(R)$. ✓

b. Projection is not commutative

$\Pi_{L_1}(\Pi_{L_2}(R)) \neq \Pi_{L_2}(\Pi_{L_1}(R))$

PAGE NO :
DATE : / /

→ In $\Pi_{L_1}(\Pi_{L_2}(R))$ L_1 must be subset of L_2 for this opⁿ to be valid. and in this case it's eq. to $\Pi_{L_1}(R)$.

→ In $\Pi_{L_1}(\sigma_C(R))$ if C uses only attr. of L_1 then we can write it as $\sigma_C(\Pi_{L_1}(R))$, else we can not.
So, in gen $\Pi_{L_1}(\sigma_C(R)) \neq \sigma_C(\Pi_{L_1}(R))$.
But. transformation:
$\sigma_C(\Pi_A(R)) \rightarrow \Pi_A(\sigma_C(R))$ is always allowed.

Note: In DBMS what happens when we run SQL query is:

SQL query (Q) → Converted to → Eq. RA query (Q_1) → Optimist → Equivalent Eff. Quer (Q_2)

→ As the data in DBMS is relation which is set of tuples. so, we'll see some set operations now.

c. Union (U)

for $A \cup B = B \cup A$, A & B must be union compatible relations.

→ Union Compatible: Two relations A & B must be having same degree (# attr.) and same domain for i^{th} attr. of both the relations.

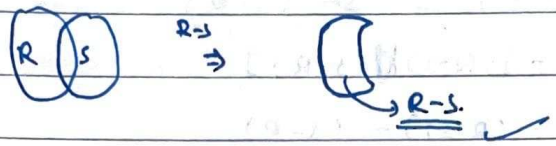
Ex: A(a,b,c) & B(e,f,g) are union comp.
iff domain of a = domain of e & d(b) = d(f) & d(c) = d(g)

i.e.

In $R(a_1, a_2 \ldots a_m)$ & $S(b_1, b_2 \ldots b_n)$ are union compatible iff:
a) $m = n$ & b) $\forall i$ domain(a_i) = domain(b_i)

PAGE NO. :
DATE : / /

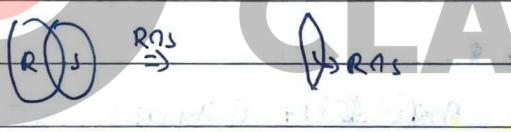
D. Set diff (−): This also req. relations to be union compatible.



In R-S we have those tuples which are in r and not in s.

$R - S \neq S - R$.

E. Intersection (∩): ~~R∪S = S∪R~~, $R \cap S \equiv S \cap R$ also req. union comp. of R & S.



In R∩S we have those tuples which are in R And in S.

(Cross)

F. Cartesian Product (×): It don't need union compatibility.

Ex: r: A B

1 2

2 3

s: X Y Z

1 1 2

7 8 9

r×s = A B X Y Z

1 2 1 1 2

1 2 7 8 9

2 3 1 1 2

2 3 7 8 9 ✓

PAGE NO :
DATE : / /

→ Intersection (∩) is not a basic operator is derived from union and diff.

$R \cap S = R - (R - S) = S - (S - R)$

$R \cap S = R \cup S - [(R - S) \cup (S - R)]$

$R \cap S = (R \cup S - (R - S)) - (S - R)$

→ ∩, ∪, − needs compatible relation but × do not need it.

→ In "×" if attr. name conflicts we use Relation · attribute convention. ($R.a, S.a$) → Preferred
or we can position no (2, 4)

→ In $S \times R$ we pair each tuple of S with each tuple of R.

$S \times R = \{tq \mid t \in S \text{ and } q \in R\}$

$|S| = x$ & $|R| = y$ then $|S \times R| = xy$
↳ Cardinality of S.

Arity $(S \times R)$ = Arity (S) + Arity (R)
↳ # attributes

→ $S \times R \equiv R \times S$ only order of attributes change. (That's all set op")

6. Rename Operations (ρ) → Rah : used to rename table or/and its columns

If $R(a, b)$ then

$\rho_{S(1,2)}(R)$ gives $S(1, 2)$ schema

$\rho_{S(1,b)}(R) \equiv \rho_{S(a \to 1)}(R)$ gives $S(1, b)$

$\rho_S(R)$ — Change table name

$\rho_{S(a,b,c,d)}(R)$ — Change table & col. name

$\rho_{(a,b,c)}(R)$ — only change of name.

PAGE NO. :
DATE : / /

Assignment Operator (←) : Used to give name to RA O/P.

Ex: myrltn. ← $\sigma_c \Pi_{a,b}(R)$

Ex: Let's have twitter dataset T(A, B) where A & B are Uid and tuple (x,y) means x follow y.
Then true friends are (x→y & y→x):

$$\Pi_{R.A,\ R.B.}\left(\sigma_{R.A=S.B \wedge R.B=S.A}\left(\rho_R(T) \times \rho_S(T)\right)\right)$$

↳ Renamed table to do Cross product with itself.

Some queries based on Twitter database: A →follows→ B

kk → mk
KK → Ak
Ak → kk
ac → mj
mj → rc

a). All people "KK" follows:

$$\Pi_{T.B}\left(\sigma_{T.A="KK"}(T)\right)$$

b). All people who follow "KK":

$$\Pi_{T.A}\left(\sigma_{T.B="KK"}(T)\right)$$

c). All pair of people who follow each other:

for this we have to combine T x T.

T_1/R		T_2/S	
A	B	A	B
x	y	y	x.

} we need such way.

So $\Pi_{T_1.A,\ T_1.B.}\left(\sigma_{\substack{T_1.A = T_2.B \wedge \\ T_1.B = T_2.A}}\right) \rho_{T_1}(T) \times \rho_{T_2}(T)$ ✓

PAGE NO. :
DATE : / /

d) people who follow atleast 2 d/t people.

⇒

T_1		T_2	
A	B	A	B
x	y	x	z

So

$$\Pi_{T_1.A}\left(\sigma_{\substack{T_1.A = T_2.A \wedge \\ T.B \neq T_2.B}}\left(\rho_{T_1}(T) \times \rho_{T_2}(T)\right)\right)$$

e) people who follow atleast 2 people:

⇒

T_1		T_2		T_3	
A	B	A	B	A	B
x	y	x	z	x	w

So. $T_1.A = T_2.A \wedge$

$T_2.A = T_3.A \wedge$

$T_1.B \neq T_2.B \wedge$

$T_1.B \neq T_3.B \wedge$

$T_3.B \neq T_1.B$ ✓

$$\Pi_{T_1.A}\ \sigma_C\left(\rho_{T_1}(T) \times \rho_{T_2}(T) \times \rho_{T_3}(T)\right)$$

f) People who follow exactly one person:

Exactly one: Atleast one - Atleast two

$\Pi_{(A)}(T)$ (d)

OR

Exactly one = 2 A

PAGE NO. :
DATE : / /

* Some derived Relational operator are: (All prev. except ∩ were basic operators)

I Join Operations:

→ Generally applying only the cross products is meaningless, i.e. rarely required. Then how to combine tables?

→ We use cross products & then selection or projection to use it, but it's inefficient to do cross product and then select as cross product will gen. many tuples from which we'll have to select. (Talk later about it).

→ Conditional join is cross product with a condition.
↳ Derived operator bcz. used very frequently.

$$\sigma_C (T_1 \times T_2) \equiv T_1 \bowtie_C T_2$$

→ Join is most common way to combine info. from two or more relations.

(Inner) Join
- Conditional Join (Theta): $R \bowtie_\theta S$, θ → condition
- Equi Join (Most Imp.): ↳ Special conditional join's where condition only has "=" operator
- ✓ Natural Join: $R \bowtie S$

PAGE NO. :
DATE : / /

→ In $\sigma_c(R \times S)$ we have huge intmdte. result but $R \bowtie_c S$ join only if condition is met.

→ Row which is not joined with any other row, is known as dangling row.

Ex:

R

A	B	C
1	2	3
4	5	6
7	8	9

Dangling Row

S

D	A
3	1
6	2

$R \bowtie_{B<D} S$ (4) is $\sigma_{B<D}(R \times S)$ (5) → In other notation

R.A	B	C	D	S.A
1	2	3	3	1
1	2	3	6	2
4	5	6	6	2

→ Schema of $R \bowtie_c S$ is same as schema $(R \times S)$.

→ Equi join is a special conditional (θ) join (old term) in which condition only contains "=".

Ex:-

$R \bowtie_{x=y} S$ is equijoin & $R \bowtie_{x>y} S$ is not.

In this also schema is unaffected.

PAGE NO. :
DATE : / /

→ Natural Join: In this we do not write condition explicitly, just equate all common attributes (same name) & write them only once in result instance.

If R(a,b,c) & S(b,c,d) then R ⋈ S (Natural join) will be (a,b,c,d).

If R and S do not have any common attribute then:
R ⋈ S = R × S.

In this schema(R⋈S) ≠ schema(R×S).

Ex:-

R

A	B
1	2
3	4

S

B	C	D
2	5	6
4	7	8
9	10	11

Dangling Row.

→ Natural join.

R⋈S

A	B	C	D
1	2	5	6
3	4	7	8

→ Equi Join

R ⋈ R.B=S.B S

A	R.B	S.B	C	D
1	2	2	5	6
3	4	4	7	8

Very Imp.
J *** Division Operation:

In mathematical algebra:

8/4 = 2 means 2*4 ≤ 8.

8/3 = 2 means 3*2 ≤ 8

So $\frac{N}{d} = Q$ → Q is largest integer such that Q*d ≤ N.

→ will be satisfied by many integer but choose the largest possible.

PAGE NO. :
DATE : / /

→ So in mathematical algebra, for int A & B, A/B is the largest integer Q such that Q * B ≤ A.

→ Similarly for Relational algebra for rel. A & B, A / B is the largest possible relation Q such that Q × B ⊆ A

For Ex:

A:	
a	b
1	1
1	4
2	1
2	2
2	3
2	4
3	1
3	3
3	4

B:
b
1
3

A ÷ B
a
2
3

} Those a's only with whom 1 & 3 both are as b.

Schema of A÷B will be (a) bcz (A÷B) * B must be union comp. with A. So it must be (a)

A÷B needs to be subset of,

a
1
2
3

} Now we'll see that with this X B ⊄ A so choose largest.

NOTE $\frac{R(A,B)}{S(A,B)}$ is an invalid division.

for R ÷ S to be successful attr(S) ⊂ (Proper subset) attr(R)
and attr(R÷S) are attr(R) − attr(S).

PAGE NO. :
DATE : / /

* → In $R(A,B) \div S(A) = Q(B)$
↳ In this we have those values of (B,) which pair with every row of S in R.

→ $\frac{(R \times S)}{S}$ is R only as every value of R is paired with each value of S in R×S.

→ If R/S = Q then Q×S is subset of R.

* Healy's Division.

→ R(kid, game), S(game) then R/S will give all kids who plays all games specified in relation S.

→ It is equivalent to (R/S) ↓
All kid - (kids who don't play some (>1) s·game)

⇒ $\Pi_{kid}(R)$ - X

Now X = those kids who don't play some s·game.

$X = \Pi_{kid}[(\Pi_{kid}(R) \times S) - R]$

↙ All kids ↙ s·games

as $(\Pi_{kid}(R) \times S) - R$ will give all those (kids, games) that are not played. and if we project only kids we'll know those kids who is not playing some game.

So $R/S = \underbrace{\Pi_{kid}(R)}_{\text{All kids}} - \underbrace{[\Pi_{kid}((\Pi_{kid}(R) \times S) - R)]}_{\text{↳ kids who don't play some game from s}}$

↳ kids who play all s·games.

So this is division opⁿ in terms of simple opⁿ.

PAGE NO :
DATE : / /

* → So, in general division in terms of other operations

$r(R) \div t(T) = Q(R-T)$

$Q(R-T) \equiv \Pi_{R-T}(r) - [\Pi_{R-T}([\Pi_{R-T}(r) \times t] - r)]$

↳ known as Healy's Division

(Think of kids & games example).

(once again: Important

$R(A,B,C,D) \div S(C,D) =$

$\Pi_{A,B}(R) - \Pi_{A,B}[[\Pi_{A,B}(R) \times S) - R]$

* Another Division Implementation: (Maier's ÷)

→ In our R(kid, game) & S(game) example:

R

kid	game
1	a
1	b
1	c
2	a
2	d
3	a
3	b

S

game
a
b

Now let k_a = Set of all kids what play game a

k_b = " " " " " " game b.

$k_a = \{1,2,3\}$

$k_b = \{1,3\}$

Now $R/S = k_a \cap k_b$

$= \{1,3\}$

PAGE NO. :
DATE : / /

R(AB) | S(B)

So R/S is eq. to $\bigcap_{\forall b_i \in B} (\Pi_A(\sigma_{B=b_i}(R)))$ for a i.e. wher Ai is set of all A values paired with bi in R.

↳ This is known as Maier's Division.

Que : 1 If |R| = n & |S| = m, min & max. value of |R ÷ S| ?

⇒ Minimum will be 0, and max can be n if m = 1 & all (R-T) attrs of R pair with that single value of S.

maximum = $\left\lfloor \frac{n}{m} \right\rfloor$

Que : 2 If |R| = n ≠ 0 and |S| = 0 then min. & max. value of |R ÷ S| ?

⇒ Anything × ∅ = ∅

From Healy's theorem:

$\Pi_A(R) - \Pi_A[(\Pi_A(R) \times S) - R]$

$\Pi_A(R) - \Pi_A[\emptyset - R]$

$\Pi_A(R) - \Pi_A[\emptyset]$

$\Pi_A(R) - \emptyset$

$\Pi_A(R)$

R/S = $\Pi_A(R)$ if S is empty

now it can be n if A in R is SK (max)

it can be 1 if A in R is single value (min)

So 1 to n.

Que : 3 If |S| = m ≠ 0 and |R| = 0 then min. and max value of |R ÷ S| ?

⇒ It will be 0.

as $|(R \div S) \times S| \leq |R|$

$x \times m \quad 0$

↳ needs to be 0 as m ≠ 0.

$\bowtie \equiv \bowtie = \bowtie_L$

$\bowtie = \bowtie = \bowtie_R$

$\bowtie = \bowtie$

PAGE NO :
DATE : / /

RA Operation.

K. **Outer join operations**: Here we include dangling rows (jinka koi nahi tha) in the final join using null.

→ Left outer join

K1 **Left Outer join**: ($\bowtie$) In R ⋈ S we include dangling tuples of R included in final result of R ⋈ S with the help of null.

K2 **Right Outer join** ($\bowtie$) In R ⋈ S we include dangling tuple of S inc. in final result of R ⋈ S. using null.

K3 **full outer join** ($\bowtie$) In R ⋈ S we inc. dangling tuple of R & S.

Ex: R =

X	Y
A	B
C	D

S =

X	W
A	F
G	H

R ⋈ S =

X	Y	W
A	B	F
C	D	null

R ⋈ S

X	Y	W
A	B	F
G	null	H

R ⋈ S

X	Y	W
A	B	F
G	null	H
C	D	null

NOTE ↔ Outer join can be natural or can be paired with some condition.

PAGE NO. :
DATE : / /

* Practice of RA: (Imp que. only).

→ $R \overset{\circ}{\bowtie}_R S$ = $R \bowtie S$ + Dangling tuples of S
→ $R \overset{\circ}{\bowtie}_L S$ = $R \bowtie S$ + Dangling tuples of R
→ $R \overset{\circ}{\bowtie} S$ = $R \bowtie S$ + Dangling tuples of S & R

} Can be paired with conditions also instead of being natural.

→ $R \overset{\text{natural join}}{\bowtie} S \equiv R \times S$ if no attribute matches.
(write in terms of basic op^n & you will get proof.)

→ $(R \bowtie S) \bowtie T \equiv R \bowtie (S \bowtie T)$ as equality cond^n only.

→ Outer join can be written in terms of basic op^n as:

$$R(r) \overset{\circ}{\bowtie} S(s) = [(R \bowtie S)] \cup [(R - \Pi_r(R \bowtie S)) \times \{(null, null, \ldots)\}]$$

(R ⋈ S): Natural tuples.
(R − Π_r(R ⋈ S)): Dangling tuples of R
{(null, null, ...)}: (s) times.
×: Ability.

Sample DB:

Instance of Sailors

Sid	cname	rating	age
22	Dustin	7	45.0
29	Brutus	1	33.0
31	Lubber	8	55.5
32	Andy	8	25.5
58	Rusty	10	35.0
64	Horatio	7	35.0
71	Zorba	10	16.0
74	Horatio	9	35.0
85	Art	3	25.5
95	Bob	3	63.5

Instance of reserves

Sid	bid	day
22	101	10/10/98
22	102	10/10/98
22	103	10/8/98
22	104	10/7/98
31	102	11/10/98
31	101	11/6/98
31	104	11/12/98
64	101	9/5/98
64	102	9/8/98
74	103	9/8/98

PAGE NO. :
DATE : / /

bid	bname	color
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

Instance of boats table.

Now some queries:

Q Names of sailors who have reserved boat 103.

⇒ Think of idea:

Sailors reserved

sid	sname	sid	bid
x	By	x	103

↳ Project this

$\Pi_{sname}(\sigma_{bid=103}(Sailors \bowtie reserves))$

↳ Natural join as sid is only common attribute.

$\sigma_{sid=sid}(Sailors \times reserves)$ is worse as cross product will form many queries for intermediate results in b/w but natural join will not. However #comparisons done here in both cross product & natural join are same.

#Comparison can be reduced by using selection before joining as:

$\Pi_{sid}(Sailors \bowtie \sigma_{bid=103}(R))$

↳ But possible query

Natural join - Reduces storage
Early selection - Reduces comparisons

PAGE NO. :
DATE : / /

NOTE ↔ Always try to filter (select) rows before joining. ALWAYS

Q: Sids of sailors who have reserved red boats.

⇒ $X = \Pi_{sid}(Reserves \bowtie \sigma_{color=red}(B))$

Q: Names of sailor who reserved red boat

⇒ $\Pi_{name}(Sailors \bowtie X)$

NOTE ↔ If we give unefficient solution query optimizer can give optimised efficient query with early use of selection & all.

→ Comparison are reduced also when we early use the projection as duplicates might get removed.

Q: Find the colors of boats reserved by "Lubber".

⇒ $\Pi_{colors}(boats \bowtie reserves \bowtie (\sigma_{name="Lubber"}(sailor)))$

Q: Name of sailors who have reserved at least one boat.

⇒ $\Pi_{name}(Sailor \bowtie \Pi_{sid}(Reserve))$

↳ It will reduce comparisons.

NOTE ↔ Read the query carefully... about what is asked, what is given, what is key, what is fk etc.

PAGE NO. :
DATE : / /

Q: Name of sailors who have reserved a green or a red boat.

⇒ Union of (Redboat sailor name, Green boat sailor name)

OR

Π_{name} S ⋈ R ⋈ ($\sigma_{color='Red' \vee 'Green'}$ (Boat))

and many other ways.

Q: Name of sailors who have reserved a red and a green boat.

⇒ ~~Name~~ Names of green boat sailors ∩ Name of red boat sailors } This is wrong.

↳ Why?

↳ As name is not key

So diff sailors having same name only may come in this intersection. So correct way is:

$$\Pi_{sname}\left(\begin{array}{c}\text{Sid of green boat sailors}\\ \cap \\ \text{Sid of red boat sailors}\end{array}\right)$$

as sid is key of sailors

$$\Pi_{sname, Sailor} \bowtie \left(\begin{array}{c}\Pi_{sid}(\text{reserves} \bowtie \sigma_{color=red}(boats))\\ \cap \\ \Pi_{sid}(\text{reserves} \bowtie \sigma_{color=green}(boats))\end{array}\right)$$

$\sigma_{color = red \wedge green}$

↳ This is nonsense as no boat is red & green.

PAGE NO. :
DATE : / /

Q: Name of sailors who have reserved at least two boats.

⇒ idea:

Sailors		Reserves		Reserves	
Sid	sname	Sid	bid	Sid	bid
↓ Project		x	y	x	z

Π_{sname} (Sailors ⋈ Π_{sid} (~~$\sigma_{R_1.bid \neq R_2.bid}$~~ (Reserves R_1 ⋈ Reserves R_2)))

$R_1.bid \neq R_2.bid$
$\wedge R_1.sid = R_1.sid$

Q: Name of sailors who have reserved exactly one boat.

⇒ Name of Atleast one − Name of atleast two

↳ Wrong as Name is not key, Two diff Karans are having 1 boat & 1 having, but he will be removed.

So

Π_{sname} (sid of atleast one − sid of atleast two)

NOTE: Take care while dealing with key or non-key attribute.

Q: Sid of sailors who have reserved all boats.

⇒ Use division here.

$\Pi_{sid,bid}$(Reserves) ÷ Π_{bid} (boats)

↳ Give sids of sailors reserved all boats.

PAGE NO. :
DATE : / /

Learn to analyse

Que: for the Relation Emp(id, age) find queries for:

a). Id of all employees whose age is not minimum.

⇒ $\Pi_{id}(Emp \bowtie_{(age > age1)} \rho_{id, age1}(emp))$

Idea.

Emp. (id, age) — Here present only if greater than some age1 ✓

Emp. (id1, age1) — Here present only if lesser than age i.e. age > age1

b) Id of all employees whose age is not maximum

$\Pi_{id}(Emp \bowtie_{age > age1} \rho_{id, age1}(emp))$

~~write~~

c) Id of emp whose age is minimum.

⇒ NI: (Id of all employees) − (option a)

d). Id of emp whose age is maximum.

⇒ NI (Id of all employees) − (Option B)

e) Id of 2nd youngest employee i.e. whose age is more than the age of exactly one other employee.

⇒ = Not minimum − age > atleast 2 people
↳ option a) ↳ T

$T_2 = \Pi_{E_1.id}(\sigma_{E_1.age > E_2.age \wedge (E_1 \times E_2 \times E_3)}$

$E_1.age > E_3.age \wedge$

$E_1.id \neq E_3.id$

PAGE NO. :
DATE : / /

→ Max. and min size of natural join of R&S can go from mn to 0 where m = |R| & n = |S|.
↳ No common attr.
↳ Common attr. but no common value.

** Relational Calculus (RC): In RA order of opn were given by us that's why it was procedural language. But RC is declarative language. So no sense in talking about efficient RC
It's of two types TRC & DRC
↳ Not for GATE

** Tuple Relational Calculus (TRC):

Here each var. is a tuple variable i.e. repr. row of relation like int. var. can hold integer, similarly, tuple var. holds tuples.

→ R(t) or t ∈ R means t can hold tuples of relation R.
→ t.A or t[A] means attribute A's value for that tuple.

TRC is written in set builder form
{t | P(t)}
↳ Condition / Predicate
↳ Set of tuples t for which P(t) is True.

There are two ways (say I/e) to write TRC, we'll solve some of the questions to understand it.

PAGE NO :
DATE : / /

Q $\sigma_{B=17}(R) \Rightarrow \{t \mid t \in R \wedge t[B]=17\} \equiv \{t \mid \exists x \in R\ (x.B=17 \wedge t.A=x.A \wedge t.B=x.B \wedge t.C=x.C)\}$

RA ↑ ; TRC way 1 ↑

Q $\Pi_A(R) \Rightarrow \{t[A] \mid t \in R\} \equiv \{t \mid \exists x \in R\ (x.A \equiv t.A)\}$

way 2 ↑ ; way-2 TRC ↑

Here x & t both are tuple var. Schema of tuple var. t is fixed on the basis of attr. seen in query.

Q $\Pi_{A,C}(R) \Rightarrow \{t.A, t.C \mid t \in R\} \equiv \{t \mid \exists x \in R[(x.A=t.A) \wedge (x.C=t.C)]\}$

known as value supplying to t.

Q $\Pi_A(\sigma_{B=10} R) \equiv \{t.A \mid t \in R \wedge t.B=10\} \equiv \{t \mid \exists x \in R\ (t.A=x.A \wedge x.B=10)\}$

not t.A.

Q. $R \cup S \equiv \{t \mid t \in R \text{ OR } t \in S\} \equiv \{t \mid \exists x \in R\ \exists y \in R\ (t.A=x.A \text{ OR } t.A=y.A)\}$

Can be taken as same var and OR b/w two. ; for minus ↑

Q. $R \cap S \equiv \{t \mid t \in R \text{ AND } t \in S\} \equiv \{t \mid \exists x \in R\ \exists y \in R\ (t.A=x.A \text{ AND } t.A=y.A)\}$

PAGE NO. :
DATE : / /

TRC2 (way 2) is somewhat result oriented we write query on the basis of what we want and nowhere we are telling how you should fetch it.

Q. $R-S = \{t \mid t \in R \wedge t \notin S\} = \{t \mid \exists x \in R \forall y \in S (t.A = x.A \text{ AND } t.A = y.A)\}$
$= \{t \mid \exists x \in R \forall y \in S (t.A = x.A \text{ AND } t.A \neq y.A)\}$

Q. $A \times B = \{(t_1, t_2) \mid t_1 \in A \wedge t_2 \in B\} \equiv \{t \mid \exists x \in A (x.a_1 = t.a_1) \wedge \exists x \in B (x.b_1 = t.b_1)\}$
$A(a_1) \times B(b_1)$

→ In TRC way 2 $x.A = t.A$ (where x is bounded tuple var) means for t.A supplier is x.A.

Q. $R(A,B) \div S(B) = Q(A) \equiv \{t.A \mid t \in R \wedge \forall x \in S \exists y \in R (y[B] = x[B] \wedge t[A] = y[A])\}$
$\equiv \{t \mid \forall x \in S \exists y \in R (x.B = y.B \wedge t.A = y.A)\}$

→ TRC Queries from our sailors and boats Database ←

Q: Names and ages of sailors with a rating above 7.

⇒ TRC 2.

$\{t \mid \exists x \in \text{sailor} (x.\text{rating} > 7 \wedge t.\text{name} = x.\text{name} \wedge t.\text{age} = x.\text{age})\}$

TRC1.

$\{t.age, t.name \mid t \in \text{sailor} \wedge t.\text{rating} > 7\}$

PAGE NO :
DATE : / /

Q: Sailor name, boat id and reservation date for each reservation.

⇒ TRC2.

$\{t \mid \exists x \in sailor\ \exists y \in reserves\ (x.sid = y.sid \wedge t.sname = x.sname \wedge t.bid = y.bid \wedge t.date = y.date)\}$

Practice PYQs of TRC (Easy)

Q Find id of emp with max salary. Emp(id, salary).

⇒ $\{t \mid \forall x \in Emp\ (x.sal \leq t.salary)\} \equiv \{t \mid t \in emp \wedge \forall x \in emp\ (t.salary \geq x.sal)\}$

NOTE $\forall x \in D\ (P(x)) \Leftrightarrow \forall x\ (x \in D \rightarrow P(x))$

$\exists x \in D\ (P(x)) \Leftrightarrow \exists x\ (x \in D \wedge P(x))$

NOTE ** $\{ \underbrace{\quad}_{\text{Only free var are used here (x)}} \mid \underbrace{\quad}_{\text{All var. other than x must be bounded.}} \}$

So the o/p tuple var. must be free and all remaining variable must be bounded.

Q: Find name of sailors who have reserved a red boat.

⇒ $\{t \mid \exists x \in sailor\ \exists y \in reserves\ \exists z \in boats\ (t.sname = x.sname \wedge x.sid = y.sid \wedge y.bid = z.bid \wedge z.color = 'Red')\}$

PAGE NO. :
DATE : / /

Q: Names of sailors who have reserved at least two boats.

⇒ $\{t \mid \exists x, y \in \text{reserves}\ \exists z \in \text{sailors}\ (x.\text{rid} = z.\text{sid} \wedge y.\text{rid} = z.\text{sid} \wedge x.\text{bid} \neq y.\text{bid} \wedge t.\text{sname} = z.\text{sname})\}$

Q. Names of sailors who have reserved all boats.

⇒ $\{t \mid \exists z \in \text{sailors}\ \forall x \in \text{boats}\ \exists y \in \text{reserves}\ (x.\text{bid} = y.\text{bid} \wedge y.\text{sid} = z.\text{sid} \wedge t.\text{sname} = z.\text{sname})\}$

Q. Sid of sailors who have reserved all red boats.

⇒ $\{t \mid \exists x \in \text{sailors}\ (x.\text{sid} = t.\text{sid}) \wedge \forall z \in \text{boat}\ \exists y \in \text{reserves}\ [(z.\text{color} = \text{Red}) \rightarrow (y.\text{sid} = x.\text{sid} \wedge y.\text{bid} = z.\text{bid})]\}$

Imp.

Unsafe query: If a TRC query provides any data which is not in the database then that TRC query is unsafe.

TRC query is safe if it provide whole data which is in database.

If o/p of TRC query is infinite then surely it's unsafe query because the database is finite.

Unsafe query ⟶ ∞ results.
(mostly (Not always))

Ex: $\{t \mid t \notin s\} \equiv \{t \mid \neg(t \in s)\}$

↳ Most famous unsafe query.

PAGE NO. :
DATE : / /

☆ Expressive power of Query Languages:

→ Each RA can be written as TRC so P(RA) ≤ P(TRC)

→ TRC is able to give unsafe query which RA can't so P(TRC) > P(RA)

Relation Algebra ≡ Safe DRCs ≤ All TRCs (Safe + Unsafe)

TRC DRC TRC DRC

↳ In terms of exp. power

→ If in $\{t \mid \_\_\}$; $t \in$ which rltn is not given then it is TRC way of query.

→ TRC query is safe iff we can write an equivalent RA query.

PAGE NO. :
DATE : / /

★★ SQL

→ Most widely used commercial (bcz. dev. by companies) relational database ~~prac~~ declarative language. (Not given by Codd).

→ Two out of 5 SQL sublanguages: (DDL, DML, DQL, DCL, TCL)

a). Data definition language (DDL): Supports creation, deletion & modification of definitions for tables and views. (Not imp. for GATE).

b) Data manipulation language (DML): Used to manipulate & retrieve rows. (Data Retrieval Queries are very imp for Gate).

→ RA & RC were theortical and SQL is practical lang. inspired from RA (less) & RC (more).

→ In theortical languages:
- Table is set of tuples
- No nulls except in outer joins
- ∃! PK ⟷ unique
- ∃ CK, ∃ SK, etc.

→ But in practical languages, we have diff req.:
- Table is multiset of tuples. (duplicate rows allowed)
- Nulls allowed (data may be absent)
- PK, CK, SK may be absent.

→ So SQL doesn't use pure relational model, also it makes sense to have such modifications.

PAGE NO. :
DATE : / /

→ SQL is not case sensitive but convention is to write keywords in Caps.

→ SQL allows duplicate tuples bcz:
- It is more flexible i.e. Real world description may need it.
- Expensive to eliminate duplicates.

→ Basic SQL query:

SELECT <attr list> — TT
FROM <relation list> — ×
WHERE <condition> — σ

* Mandatory in each query
Optional*

→ write "distinct" after select to remove duplicate tuples, by default not removed.

NOTE ↔ In SQL (in RA):
If AB is PK then no null in A&B and no duplicates in AB.

→ Let R(A, B) & S(C, D)
then

SELECT A, D FROM R, S ≠ $\Pi_{A,D}(R \times S)$ = SELECT DISTINCT A, D FROM R, S

but,

SELECT A, C FROM R, S = $\Pi_{A,C}(R \times S)$

PAGE NO. :
DATE : / /

Conceptual

→ Evaluation of basic query happens as: (Not actual)

1) FROM i.e. Cross product
2) WHERE i.e. Condition
3) SELECT i.e. Select attributes

} Read & write query in this order.

→ SELECT * gives all attributes, * is wildcard.

→ To print table R, query is not R, it is:

SELECT * FROM R ≡ R
↳ SQL ↳ In RA

→ In SQL retrieval query original database doesn't change the query o/p is bring in main memory

→ SELECT name, age FROM stud.
SELECT stud.name, stud.age FROM stud
SELECT name, age FROM stud s
↳ called alias.
SELECT s.name, s.age FROM stud s
SELECT name, s.age FROM stud s
SELECT stud.name, s.age from stud s } X
↳ This is not allowed

Once alias is given then can't use table name.

→ We donot know how actually SQL is evaluated i.e. in which order as SQL is dec. language it is decided by DBMS evaluator, but we do our own conceptual evaluation of SQL query (in order told above), answer will obviously be same.

PAGE NO. :
DATE : / /

→ Our conceptual evaluation order is the most inefficient way to compute query. DBMS is smart it has efficient strategies, it uses it.

Q: Name and age of all sailors.

⇒
```
SELECT sname, age
FROM sailors.
```

→ Distinct keyword works on whole tuple like (sname, age) if used as

```
SELECT DISTINCT sname, age
FROM sailors.
```

Q: All sailors with rating > 7.

⇒
```
SELECT *
FROM sailors
WHERE sailors.rating > 7.
```
(sailors ↓ optional here)

⇒ ~~Alias~~ Aliases can be given with AS keyword, but it is always an optional keyword.

FROM sailors S ≡ FROM sailors AS S.

Range vars/

→ Aliases are good for reading & becomes mandatory when we use same table more than once.

→ Once alias is written table name cannot be used for attr. now.

In SQL <> Not equal to

PAGE NO. :
DATE : / /

for column

→ Aliases can also be used with SELECT clause as a headers

Ex: SELECT sname AS 'Sailor Name'
↳ optional here too.

≡ SELECT sname 'Sailor Name', age FROM Sailors

→ As can be used optionally to Rename Relation or to Rename column headers.

Q: ~~Sids~~ snames of sailors who have reserved a Red boat.

⇒ SELECT sname
FROM Sailors S, Reserves R, Boats B
WHERE B.color = Red AND
B.bid = R.bid AND
R.sid = S.sid.

Q: Colors of boat reserved by Lubber:

⇒ S b.colors
F boat B, sailor S, reserves R
W B.bid = R.bid And S.sid = R.sid AND Sname = "Lubber".

NOTE ↔ Use idea of drawing tables side by side to know the requirement.

PAGE NO :
DATE : / /

★ Set Operations in SQL:

If the result of two queries are union compatible then we can do:

a) UNION
b) INTERSECT
c) SET DIFFERENCE (Called EXCEPT or MINUS)
↳ std. ↳ In oracle DBms

} Exactly same as RA.

*
→ Result of above operation do not contain duplicated. So, duplicates are eliminated from the list.

→ To retain duplicates we use ALL keyword with this operation. (Frequency)
i.e. UNION ALL, INTERSECT ALL and EXCEPT ALL.

→ Ex: R

A
1
2
2
2
3

S

A
1
2
2
4
5

X R union S is invalid query actually written as

(SELECT * FROM R) UNION (SELECT * FROM S)
↳ X ↳ Y

So X UNION Y =

A
1
2
3
4
5

↳ for this, remove dup. from i/p & then from o/p.

PAGE NO. :
DATE : / /

X Union All Y = A

1
1
2
2
2
2
2
3
4
5

No duplicate is removed.

X INTERSECT Y = A

1
2

↳ Remove dup from i/p & o/p.
↳ not needed.

X INTERSECT ALL Y = A

1
2
2

Select using counting i.e. for each common write once even if duplicate.

X EXCEPT Y = A

3

↳ Remove dup. from i/p.

X EXCEPT ALL Y = A

2
3

Use counting method. like two 2's are removed but one '2' left.

FROM ≠ from
'From' ≠ 'from' } In SQL

PAGE NO :
DATE : / /

Q: Sids of sailors who have reserved red as green boat.

→ Q1 → S sid
F ~~Sailors~~ Reserves R, boat B
W (reserves.sid = b.bid) AND
(b.color = 'green' OR b.color = 'red').
↳ string, case sensitive

Q2 → query for rid of green boats
UNION
query for rid of red boats
} Works bcz bid is key and used OR
→ UNION ALL we also give same result as bid is key

Q: name of ~~sailors~~ sid who have reserved red AND green boat.

~~Here above boats queries~~

Q1 → S bid
F R, B
W R.bid = b.sid AND
b.colr = 'green' AND b.color = 'red'

WRONG
↳ will always give an empty relation as no boat is both red and green

Q2: query for rid of green boats reserved
INTERSECT (ALL)
query for rid of red boats reserved
} ↙

Correct query.

PAGE NO. :
DATE : / /

Q: What if asked sname instead of sid in prev. both ques'ns.

⇒ In green OR red query sname directly will work but in green AND red it'll not directly work

Can't intersect like:

sname of green boat
INTERSECT
sname of red boat

} Wrong (may be same name diff. people here)

sname { sid of green INTERSECT sid of red } } Correct. ✓

** So take care of (non-key) $\overline{\cap \cup}$ (non-key)

Q: sids of sailors reserved red boats but not green boat

⇒ Sids of red boats
EXCEPT
sid of green boats.

S sid
FROM R, B
W R.bid = B.bid AND
B.color = 'Red' AND B.color <> 'Green'
(translation here)

} Wrong (Gives all red boat reservers) (may or may not green)

PAGE NO. :
DATE : / /

Q: names of sailors booked red but not green boat.

→ Sname of red
EXCEPT
Sname of green } X

Get sids first using except and then get sname.

Q: sid of sailors who have not reserved red boats.

⇒ All sids − (Red boat sids)
↳ from sailors.

WRONG

S sid
F R, B
W R.bid = B.bid AND B.color <> "Red".

It will give those sids who have booked some non-red boat.
↳ No matter if also booked red

Q: All pairs (distinct) of same color boats (dif. boat of same color)

⇒
```
SELECT DISTINCT b1.bid, b2.bid
FROM boat b1, boat b2
WHERE b1.color = b2.color AND
      b1.bid ≠ b2.bid.
```

↳ It will give distinct pairs like 101, 102 once only
102, 101 ↳ absent

Let 102, 101 bid have same color

Idea

b1		b2	
bid	color	bid	color
X	a	X	a

a ≠ a → =

If used <, then (101, 102) both tuples (102, 101) present

PAGE NO. :
DATE : / /

→ In SQL we don't have ∧, ∨, ¬, ≠ we have AND, OR, NOT, <>. resp.

→ SELECT ≡ SELECT ALL (by default ALL present)
UNION ≡ UNION DISTINCT (by default distinct present).

→ SELECT clause may also have arithmetic op^n like +, −, *, and / to show some change attr. However actual DB ~~data~~ don't change

Ex:

SELECT id, Salary * 1.1 from employee
↳ Promoted salary may be.

→ Constant in an exp. is also allowed in SQL

Ex:

SELECT id, salary, 'Rs.' As currency FROM employee.
↳ Constant attribute } giving illusion of currency column.

→ SQL keywords only are case insensitive but things are case sensitive.

→ WHERE clause applies on one tuple at a time, independently.

→ In cross product by default duplicates are present i.e. |R| = n & |S| = m then |R,S| = n×n for sure.

→ In GATE minor issues like sensitivity and all are not of much importance. focus on meaning of query.

PAGE NO. :
DATE : / /

** SQL Aggregate Operators:

* Assume non-null relations for now *

→ Aggregate functions are those functions that taked a collection of values as i/p and return a single value. There are five built-in aggregate functions in SQL:
avg, min, max, sum, count.

→ These are generally used to summarise the table.

→ MIN and MAX will result same even if distinct attribute is given. i.e MIN(A) ≡ MIN(DISTINCT A). ✓
But COUNT, AVG and SUM will eliminate duplicate tint if DISTINCT is used.

Ex:

A	B
1	3
3	4
3	2

Sum(A) = 7
Sum(DISTINCT A) = 4
Count(A) = 3
Count(DISTINCT A) = 2

Max(A) = 3
Min(A) = 1
AVG(B) = 3
AVG(DISTINCT A) = 2
(3+1)/2 = 2 ↗

→ Count(*) is a special function which counts # tuples in the relation. Count(A) & Count(*) will give same o/p if no null values in A.

→ Count(DISTINCT *) is an invalid clause.

PAGE NO.:
DATE: / /

→ ALL and DISTINCT are basically opposite of each other
ALL is present at all places by default except at UNION, INTERSECT and MINUS.
So Count(A) ≡ Count (ALL A)

→

Domain of Attr.	Req.	function
Numeric	Only.	Sum
Numeric	only.	Avg.
Lexiographically String / Numeric		Min
String / Numeric		Max
Any Domain.		Count

Q: Avg. age of sailors with rating of 10.
⇒ S avg(age)
F S
W S.rating = 10.

NOTE ↔ Select sid, max(age) from sailors; is a WRONG query. (See note below)

Q: Find

NOTE ↔ If we use atleast one aggregate function in SELECT clause then NO Unaggregate attribute is allowed in select unless there is a GROUP BY clause.

Q: Find name and age of oldest sailor.
→ Select name, age FROM sailors
WHERE age = max(age);
↳ can't get max of whole table
valid but WRONG o/p
Why?
↳ because where is applied on one tuple at a time.

Done using NESTED query. Told later.

PAGE NO. :
DATE : / /

```
SELECT  name, age
FROM    sailor
WHERE   age = ( SELECT max(age) FROM sailors )
```

Here nested query will not give value but a whole relation like | age | 50 |, still age will be able to compare it.
This is special allowed case.

NOTE: null (op) anything = null

Ex: null + 10 = null ; null * 20 = null ; etc

★ Aggregate functions in case of NULL values:

Aggregate function ignores the null value except Count(*).

Ex: Sum(A) = 60

A
10
20
null
10
null
20

However 10 + 20 + null + 10 = null but aggregate function ignores it.

Count(A) = 4 Min(A) = 10 Avg(A) = 15
Count(*) = 6 Max(A) = 20

→ Just ignore null except for count(*) ↳ Alag hai

→ Before calculating remove nulls.

PAGE NO. :
DATE : / /

→ If A is an empty relation then:

Count(A) = 0 , min(A) = max(A) = Sum(A) = Avg(A) = null

→ Let A

null

null

count(A) = 0

count(*) = 2

Remaining Ag = null

Que: Guess the o/p?

SELECT sid FROM Sailor WHERE age = max(age)

⇒ All sids will be printed. as max(age) will get only current tuple age.

** Null values in SQL.

There are many interpretations that can be put on null values. Some of them are:

a) Value unknown
b) Value unapplicable

→ Arithmatic opⁿ with null gives null.

→ Comparison with null is not null, it is unknown.

→ In SQL, any truth variable can have three values (like 2 we had traditionally):

- True
- False
- Unknown

PAGE NO :
DATE : / /

Many desc.

→ Unknown is diff from null. Unknown is a Truth value.

→ NULL <comparison op> anything = Unknown.

WHERE clause passes a tuple only when "True". It do not allows when "false" or "unknown."

for Ex: R2

A	B	O/P
Null	Null	Unknown
3	3	True
2	1	false
3	Null	Unknow
Null	4	Unknown

SELECT * FROM R
WHERE A-B=0

O/P = R2

A	B
3	3

→ $F \vee F = F$; $T \vee F = T$; $T \vee U = T$; $F \vee U = U$; $T \vee T = T$
$F \wedge F = F$; $T \wedge T = T$; $T \wedge U = U$; $F \wedge U = F$; $T \wedge F = F$
$\neg T = F$; $\neg F = T$; $\neg U = U$, $U \wedge U = U \vee U = U$

Where U is unknown.

Ques: How to find # null values in an attribute.

⇒ SELECT COUNT(A) FROM R where A= null ;
↳ This will eliminate already AND ↳ NOT Allowed

→ SQL do not allow NULL as operant. To compare it give keyword
"IS NULL" and "IS NOT NULL"
↳ single keyword.

SELECT COUNT(*) FROM R WHERE A IS NULL.

PAGE NO. :
DATE : / /

→ In SELECT * FROM R WHERE A>10 OR A<=10.

↳ True when A is not null
↳ Unknown when A is null

so it will give all ~~tuples~~ tuples where C is not null.

* Exception in NULL: null = null is True } Anomaly

~~Duplicate~~.

A	B	
null	null	Duplicate ✓
null	null	
null	1	Duplicate ✓
null	1	Different ✗
2	null	

→ Anomaly as null = null must be unknown but it's True. so Duplicate ✓

} Human Eye wise.

null = null must be unknown but it is TRUE.
(comp. Op)
null # anything = unknown

except, null = null = True. ✓

→ To disallow null values in table we specify NOT NULL while defining that field in DDL. It is implicit constrain for PRIMARY KEY constraint.

Q: Find sum of all salaries, max., min and avg salaries and distinct pays of all employees

→ SELECT SUM(sal), MAX(sal), MIN(sal), AVG(sal), COUNT(DISTINCT sal) FROM employee;

PAGE NO :
DATE : / /

Q: Retrieve id of all employees who don't have supervisors.

⇒ SELECT id FROM emp
WHERE sup_id IS NULL;

~~sup_id = null~~

why sup_id = null is WRONG? ⇒ Bcz. SQL donot allow it.

WRONG Reason.
- It will always give Unknown as O/P
- i.e. when sup-id=10 then 10=null → Unknown
- and so, empty Relation

★★ Grouping of tuples: (Group by and Having clause)

Consider Relation: Students (id, name, state, marks)

Q: Avg. marks of all students

⇒ Select Avg(marks) FROM Students.

Q: Avg. marks of student from state Goa.

⇒ Select Avg(marks) FROM Students WHERE State = "Goa".

Q: Avg. marks state wise.

↳ For such situation i.e. to create groups & then its summary we need Group by clause.

Here we need to imaginary group tuples.

SELECT State, Avg(marks) FROM Students
GROUP BY State.

↳ Creates imaginary groups of states.

PAGE NO. :
DATE : / /

→ In SQL-92 it is not allowed :

SELECT avg(marks) from stud GROUP BY state
↳ state mi minghere

↓
Allowed in practical DBMS s/w but not in SQL 92 std.

*→ Any exam follows SQL-92 standard.

Q. Give avg. marks state wise but only for those states where avg. marks > 5.

⇒ Here we need to filter groups and for filtering groups we use Having clause. We can filter after executing group by ; so we having after group by.

```
SELECT   state, AVG(marks),
FROM     student
GROUP BY state
HAVING   Avg(marks) > 5;
```
↳ filtering groups. ✓

→ WHERE clause is used to filter tuples and HAVING clause is used to filter groups. (independently, one after another).

→ Aggregation fun is most useful with Groupby clause. Groupby is almost useless w/o aggregation.

→ When we use group by clause and then we aggregate function in SELECT clause then aggregation happens within group.

PAGE NO. :
DATE : / /

→ Without groupby clause, select imagines whole relation as a single group and then does aggregation. ✓

Qu: Predict the o/p:

R(A, B)

A	B
1	2
1	3
5	7
1	3
6	7
null	null
null	null
6	null

a) Select A, Sum(B)
From R
Groupby A;

O/P:

A	Sum(B)
1	8
5	7
6	7 → Not null
Null	Null

4 groups created

b) Select A, Count(A)
FROM R Groupby A;

A	Coun(A)
1	3
5	1
6	1
null	0 → Note this.

c) Select A, Count(Dis A)
FROM R GB A;

A	Count(A)
1	1
5	1
6	1
null	0

NOTE: If groupby is used then almost one tuple per group in output. ✓ (due to having ↗)
(Exactly one if NO having clause)

PAGE NO. :
DATE : / /

D. Select A from R Groupby A;

⇒

A
1
6
5
null

} Groupby is almost useless w/o aggregation

→ Groupby w/o Aggregation is equivalent to select distinct.

→ We can groupby by more than one attr. also but it is invalid in SQL 92 std.

E ⇒ SELECT A, B, sum(B) from R Groupby A } Invalid query.

**NOTE Order for Conceptual Evaluation:

1) FROM
2) WHERE
3) GROUP BY
4) HAVING
5) SELECT
6) ORDER BY

→ If groupby X then this X must be in select and X can be one attr. only, —By SQL 92 std.

→ SELECT X, Y, Z, (must be aggregated.)
groupby X } where X, Y, Z are either in groupby or are aggregated.

→ Separate group of null is created by groupby and due to anomaly each null is treated same.

PAGE NO. :
DATE : / /

→ Select X
Groupby Y
} If X & Y are set of attr. and both are unagregated then $Y \subseteq X$ (i.e. $y=x$ by SQL-92).

→ #tuple in o/p ≤ #groups , if groupby is used.
#tuple in o/p = #groups , if groupby w/o having is used.

→ Having X , Here X is either aggregate any attribute or subset of groupby attributes.

Ex: Groupby AB
Having A>10; (Avg(B)>10 ; etc.) <>0 ✓ Invalid

→ W/o groupby we can not have Having clause.

Q: Age of youngest sailor for each rating level.

~~Select (Age) from Sailor~~

⇒ SELECT min(Age), Rating FROM Sailor
GROUPBY rating;

* Common mistake while using "Having" clause.

Q: Age of youngest sailor for each rating level above 7.

⇒ Here we can filter both tuple as groups (but NOT always)

Q1: Select min(Age), Rating FROM Sailor WHERE rating > 7
GROUPBY rating ;

Q2: SELECT min(Age), Rating FROM Sailor ~~WHERE~~
GROUPBY rating HAVING rating > 7 ;

Here Q1 ⇔ Q2 ✓

PAGE NO. :
DATE : / /

As sailor with rating ≤ 7 do not affect our grouping so we can eliminate them before even making groups.

Q: Suppose that we want to count # emp. whose salary exceed $ 40000 in each dept. but only for dept. where more than 5 emp work.

⇒ This query is wrong here bcz we can't eliminate less than 40k workers before creating groups.

```
SELECT dno, count(*) FROM emp
WHERE salary > 40000 GroupBY dno
HAVING count(*) > 5;
```

It will give only those dept. where >40k salary emp. are more than 5.

So, be careful

☆ Order By Clause:

ORDER BY <list-of-attr> <ASC/DESC>. (default)

↳ used to sort the tuples on the basis of attr. list given

→ By default order is Ascending given by ASC.

→ DESC keyword can be used for getting o/p in descending order. ✓

PAGE NO. :
DATE : / /

→ Order BY A, B, C means order by A, if same then consider B, if same A B then consider C.

→ This clause is applied at last of the query and also conceptually evaluated at last.

Trick → If in query no having clause is there is groupby is asked #tuples then just give distinct attribute count.

→ Remember GATE answer are based on SQL-92 standard, so memorize such rules only.

☆ Nested Queries

↓

→ Query within Query

→ Query which is part of another query is called a subquery

→ Groupby and Orderby cannot contain subquery as they expect list of attributes only.

→ Subquery is allowed at SELECT iff that subquery gives a single tuple with a single attribute (possible w/ agg. w/o groupby) It is treated as constant attr. the way were using "2" in salary.

→ Query which is main i.e. which contains subquery is called outer query.

PAGE NO. :
DATE : / /

There are two types of subqueries. Does subquery uses attributes of outer query ? (Dependent on outer query ?)

- Yes → Correlated Subquery
- No → Simple Subquery.

→ Simple subquery is evaluated once only and then the result is used in outer query.

→ Correlated subquery is recomputed for every row of outer query table.

→ SELECT, FROM, WHERE, HAVING can have subquery
told when — only Correlated Subquery. — most common

→ Some SQL keywords for set comparison

a) IN (∈), NOT IN (∉) : Set membership keyword

S IN R where S is tuple & R is Relation
↳ True iff tuple S is in R else false.

for all we study now
Here R needs to be some subquery, it can not be Relation directly from DB.

Ex: S IN Sailors is wrong ✗
S IN (SELECT * FROM Sailors) is correct. ✓

S IN subquery
↳ Mandatory

PAGE NO :
DATE : / /

Q: Name of sailors reserved some boat

⇒ SELECT name FROM sailors WHERE
sid IN (SELECT sid FROM reserves)

- sid → sid of sailors → Scoping Rule. ✓ → An attribute belongs to closest surrounding relation which has this attribute.
- Must be compatible also
- Subquery Mandatory → simple subquery → Computed once. ✓

Q: Name of sailors who have reserved boat 103.

⇒ SELECT name FROM sailors WHERE
sid IN (SELECT sid FROM reserves WHERE bid = 103);

Q: Name of sailor who have not reserved boat 103.

⇒ SELECT name FROM sailors WHERE
sid NOT IN (SELECT sid FROM reserves WHERE bid = 103); ✓

~~HT~~

SELECT name FROM sailors WHERE
sid IN (SELECT sid FROM reserves WHERE bid <> 103);

This will give those sailors who have booked any non 103 boat.

Q: Names of sailors who have reserved red boat.

⇒ SELECT name FROM sailors WHERE
sid IN (SELECT sid FROM ~~boat~~ reserves WHERE
bid IN (SELECT bid FROM boat WHERE
color = 'red')));

PAGE NO.:
DATE: / /

→ So, we can have multiple level nested query.

→ for conceptual evaluation solve the simple innermost query & then go up.

Q: Those sailor name who have either not reserved any boat or reserved Red boat only. i.e. never reserved Non-red boat.

⇒ SELECT sname FROM sailors WHERE sid NOT IN
(SELECT sid FROM reserves WHERE bid NOT IN
(SELECT bid FROM boats WHERE color = 'red'));

→ SQL internally converts single attr. single tupple relation to single value.

for ex:

SELECT sid FROM Stud WHERE
marks ≥ (SELECT max(marks) FROM stud)

↳ guarantees single attr. single tuple
So valid

It will give sid of topper.

Subquries we have seen till now are simple subqueries, now we will see correlated subquries.

↳ Here, inner subquery could depend on the row that is currently being examined in the outer query so, we need to compute inner query with each tuple of outer query.

PAGE NO :
DATE : / /

For Ex:

```
SELECT * FROM R
WHERE A > (SELECT COUNT(*)
FROM S WHERE R.B = S.B)
```

↳ Makes it correlate query

R(A,B)

A	B
1	2
3	4
5	6

S(B,C,D)

B	C	D
2	4	6
4	6	8
4	7	9

It's o/p:

when R.A = 1 then inner query gives 1 , 1>1 → false ✗

R.A = 3 " " " " 2 , 3>2 → True ✓

R.A = 5 " " " " 0 , 5>0 → True ✓

So o/p is

R

A	B
3	4
5	6

This is how we evaluate correlated nested query.

* EXISTS , NOT EXISTS : EXISTS R gives True iff R is non-empty.

↳ To check emptiness of relation

EXISTS <Subquery>

↳ can't be relation directly.

↳ True iff subquery returns atleast 1 tuple

NOT EXISTS <subquery>

↳ True iff subquery returns 0 tuple else false

PAGE NO.:
DATE: / /

Ex: Select * FROM R
WHERE EXISTS (Select *
FROM S WHERE B<C)

R =

A	B
1	2
3	4
5	6
7	8

S =

C	D
2	10
4	12
6	14
8	16

B	Inner query o/p.	WHERE
2	4 6 8.	Pass
4	6 8	Pass
6	8	pass
8	Empty Relation	fail

So o/p is

A	B
1	2
3	4
5	6

→ Any aggregate function will give exactly 1 tuple per group (null possible) but empty row nor.

Ex: R → any agg. fun on R will give non-empty relation.

Ex: for above R(A,B) & S(C,D)

SELECT * FROM R WHERE
EXISTS (SELECT count(*) FROM S WHERE B<C);

EXISTS will be True always; count(*) → This will never give empty relation (can be 0, null, any int but will be present)

So o/p: is R.

PAGE NO. :
DATE : / /

→ "NOT EXIST (select <Aggregate> _ _)
will always return empty relation as NOT EXIST
can never be True.

Q: write a correlated query to find the names of sailors who have reserved boat 103.

⇒ Select S.sname FROM sailors S
WHERE EXISTS (SELECT * FROM Reserve R
WHERE R.bid = 103 AND
R.sid = S.sid) ;

* Unique, Not unique: When we apply unique to a subquery it returns true if no row appears twice in the answer to the subquery i.e. no duplicates
Also, True is returned if Relation is empty.

∀ ∃
* All, Any | Some:

ANY|
<op> ALL subquery
↓
comparison

Ex: S ≥ ALL (subquery)
↳ True iff S ≥ all of the values from subquery

S ≤ ANY (subquery)
↳ True if any one value ∃ S from subquery

PAGE NO. :
DATE : / /

→ If empty set then ALL returns True like ∀.

→ If empty set then ANY returns false like ∃.

Q: Find sailors whose rating is better than some sailor called Horatio.

⇒ SELECT * FROM SAILORS
WHERE rating > ANY (SELECT rating FROM sailor WHERE name = "Horatio"); ✓

Q. Find the sailors with the highest ratings.

⇒ Q1 : SELECT * FROM SAILORS
WHERE rating >= ALL (SELECT rating FROM sailors);

Q2: SELECT * FROM SAILORS
WHERE rating = (SELECT MAX(rating) FROM sailors);

→ S = ANY R is eq to S IN R
S <> ALL R is eq to S NOT IN R.

→ Take care of exists with aggregate function.

Q: 2D Names of sailors who have reserved all boats.

⇒ Reserve (sid, bid) ÷ boats (bid) = Ans

↳ This in SQL

Select sid=x if NOT EXIST (All boats except booked by sid=x)

PAGE NO. :
DATE : / /

⇒ SELECT sid FROM Reserves R1
WHERE NOT EXISTS (SELECT bid FROM boats
EXCEPT
SELECT bid FROM reserves R WHERE R.sid = R1.sid)

Another idea :

Those sids of sailors for which not exist a bid such that not exist record in reserves table with x, y value

⇒ SELECT sid FROM Sailors S1
WHERE NOT EXISTS (SELECT bid FROM boats B1
WHERE NOT EXISTS (SELECT * FROM
Reserves R1 WHERE
R1.bid = B1.bid ∧
R1.sid = S1.sid))

Ques : In SQL, relations can contain null values, and suppose comparison with null are treated as false. Which of the following pair is not equivalent ?

a. x = 5 not (not (x = 5)) c. x ≠ 5 not (x = 5)
b. x > 4 & x < 6 , x is an int d none of the above.

⇒ In c is x = null
null ≠ 5 = false
NOT (null = 5) = Not (F) = True.
} Not eq.

← Practice Pyqs →

Module-3
ER model

PAGE NO.:
DATE: / /

The general flow of DB design process is:

A) Requirement Analysis (Interaction with user)
↓
B) Conceptual Design (ER model) → to model actual DB
↓
C) Logical Design (Relational model)
↳ Schema Rel. + FD.
↓
d) Implementation

→ So ER model is one of the conceptual model designed before logical design made. No DBMS is based on ER model, they are based on relational mode; so we convert ER model to relational model.

→ ER model and relational model must be treated diff.

→ Most DBMS use relational model and ER model is famous model to conceptually design actual RDB.

→ ER model - By peter chen in 1976.
Relational Model → By EF codd in 1971.

* Entities and Entity Set: An entity is anything that can be uniquely represented
Ex: product, an emp, a s/w defect etc.

→ Each entity has a set of attributes & entities may be uniquely identified by some set of attributes (not necessary). Each attribute has a name, domain & corresponding value for that entity.

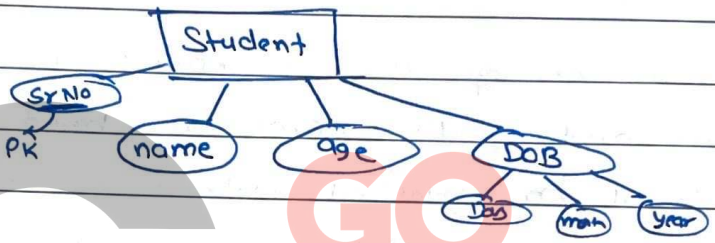
PAGE NO. :
DATE : / /

→ Entity set is a named collection of entities of same type with the same attributes.

Every entity in the entity set has the same set of attributes. and every entity in the entity set has its own value for each attribute.

→ Attribute of entity set are repr. by an oval and Entity set is repr. by rectangle.

Ex:

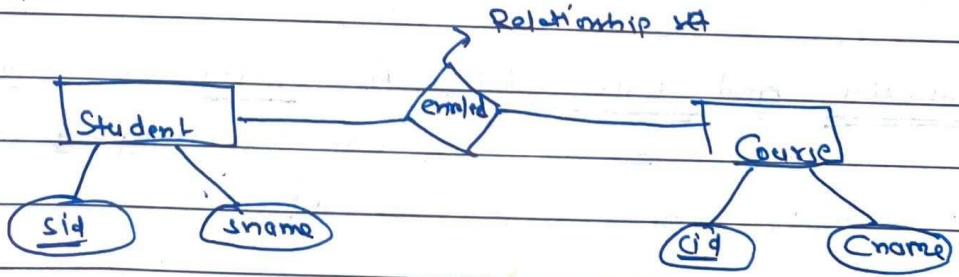


→ In ER model also we can have PK, CK & SK.

*→ Tuple of Rltnal model is Entity & Table of RM is Entity set in ER model.

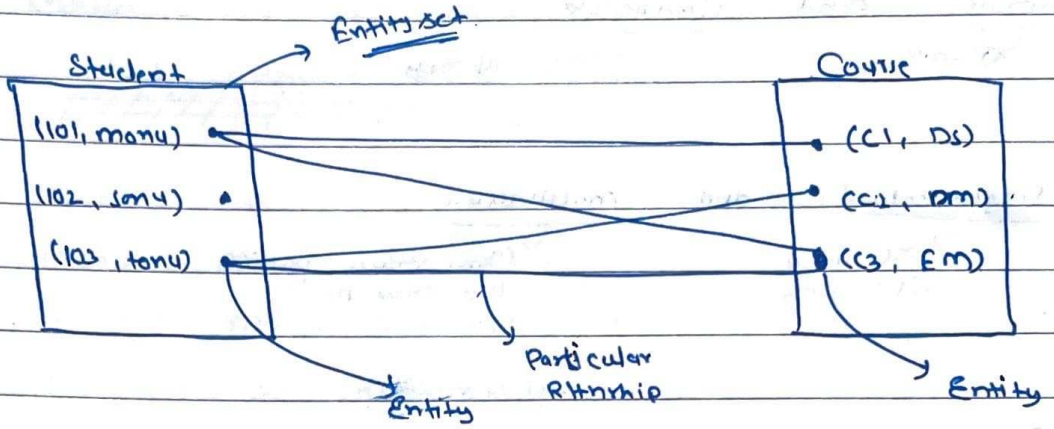
* Relationships: It is the association b/w entities Given by diamond ◇.

Ex: My



PAGE NO. :
DATE : / /

Visualization:



→ Relationship set is the set of all similar relationships b/w same entity types.

It is a mathematical relation involving n-entity set, $n \geq 2$.

If $E_1, E_2 \ldots E_n$ are entity sets, then relationship set R is a subset of:

$$\{(e_1, e_2 \ldots e_n) \mid e_1 \in E_1, e_2 \in E_2 \ldots, e_n \in E_n\}$$

↳ Relationship of degree n.

→ Relationships can also have attributes called descriptive attributes, they describe a particular relationship.

Ex: Relation b/w s/w defect and developer can have attribute date_assigned.

→ Degree of relationship set is # entities involved in relationship. (Mostly binary)

PAGE NO :
DATE : / /

→ There are various types of attributes in ER model:

a) Simple and Composite
   - Simple → Atomic
   - Composite → have subparts

b) Single valued and multivalued
   - Single valued → one value like DoB
   - multivalued → Many values possible like phone no.

c) Base and Derived.
   - Base → store in DB like DoB
   - Derived → Derived from other attribute like age

Ex:
- Simple Single valued : age, password
- " multi valued : phone no, hobbies
- Composite single valued : name
- " multivalued : address
- Base Attribute : Pincode, DoB
- Derived Attribute : state, age.

→ PK / CK / SK are called identifying attribute and have underline under their name.

→ We can also represent various constraints in ER model
- key constraints (by underline)
- mapping cardinalities (1 to 1, 1 to many, etc)
- participation constraints (partial or full).

PAGE NO. :
DATE : / /

→ also called Cardinality ratio.

* **Mapping Cardinalities / Types of Relationships**: It tells how many entities can be ass. with an entity via a particular relationship set.

A. 1 to 1 mapping (or relationship):

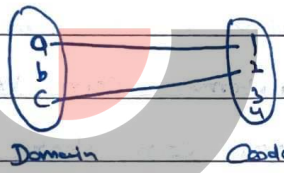
B. Many to Many Relationships.

C. One to Many Relationships

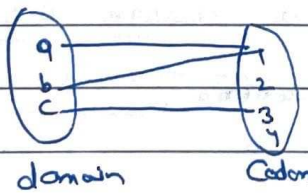
D. Many to one Relationships.

* **Types of Participation.**

→ **Partial function**: Every element of domain is mapped to **at most** one element of codomain.
↳ Optional Participation



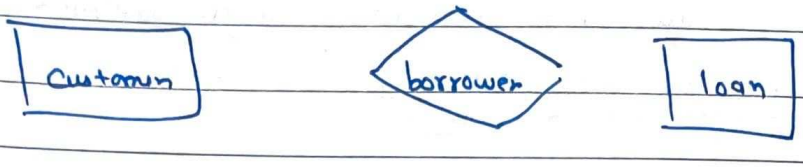
→ **Total function**: Every element of domain is mapped to **exactly** one element of codomain
↳ Mandatory Participation



Total fun → Partial fun
↳

PAGE NO :
DATE : / /

Ex: Borrower relationship b/w customer and loan



borrower can be:

a) One-to-one mapping: Each customer can have only 1 loan & can't share it with anyone else like spouse

b) One to many mapping: One customer can have many loans but each loan has one customer only i.e. loan is not shared.

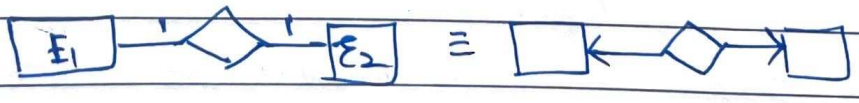
c) Many to one Relationship: One customer can have one loan & customers can share loan. i.e. many customers having one loan.

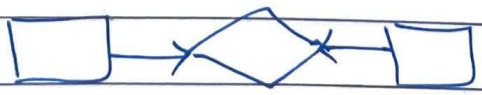
d) Many-to-many Relationship: One customer can have many loans & loan can be shared.

↳ best handles real world scenario

* Repr. of various mappings:

a) One to one.





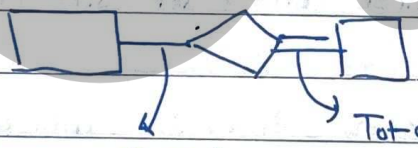
PAGE NO. :
DATE : / /

b. Many to many:

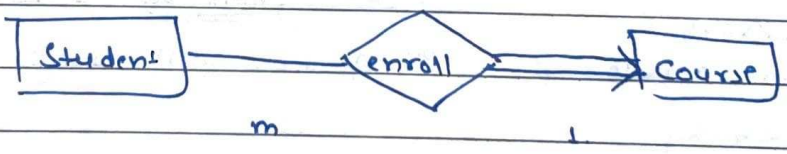
c. One to many:

D. Many to one:

* Remember NOTE: Arrow in a relationship points towards 1- side of relationship.



Ex:



→ Each course has atleast one student due to full participation of course.

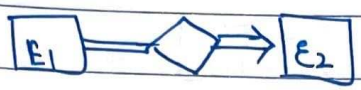
→ Student may not be enrolled in any course due to partial participation of student.

→ Many students can be in ~~some~~ one course. due to many to one relationship.

→ Student can't enroll in more than one course.

PAGE NO. :
DATE : / /

Ex:2 Comment on $|E_1|$ and $|E_2|$



$\Rightarrow |E_1| \geq |E_2|$ ✓

→ Relationships might not be identified by its attributes, but is identified by the entities participating in the relationship.
Like In Ex: ny2 R_1 may be $(101, C_3)$
R_2 " " $(103, C_2)$
R_3 " " $(103, C_3)$
sid cid. ✓

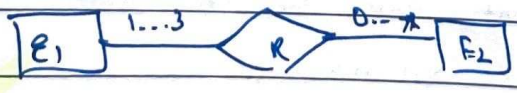
→ Entity set has its own PK/CK, if not then that entity set is called weak entity set (see later)

→ In 1-1 Relationship, PK of E_1 or E_2 will work as PK of relationship

→ In 1-m relationship, PK of E_2 will work as PK of relationship. (many side)

→ In m-m relationship PK of (E_1, E_2) together will work as PK of relationship.

→ Another repr. (numerical)



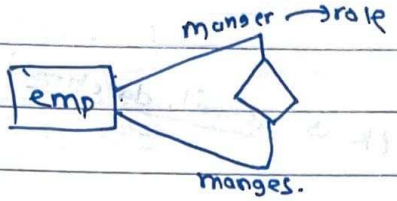
Entity of E_1 can associate from 1 to atmost 3 entity of E_2

Entity of E_2 can associate from 0 or more entity of E_1. ✓

PAGE NO. :
DATE : / /

* Self-Refrential Relationship: (Recursive / cyclic).

Ex:



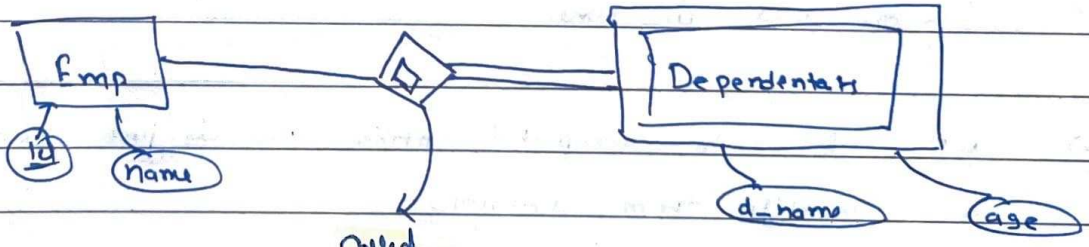
→ Entities have roles in relationships. It can be ambiguous in self ref. relationship.

* Week - Entity Set.: Entity set which do not have any identifying attribute. ie w/o any key.

Existence of weak entity set depends on existance of Owner entity ~~(Owner of it)~~.

Repr. by double rectangle in ER model.

Ex:



Called identifying relationship → to identify two weak entity uniquely.

Owner of WES (Strong) — Week Entity set

NOTE: Owner ES can be weak a strong

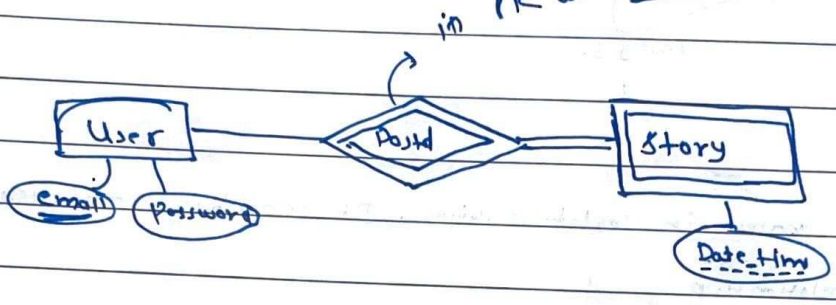
Ex:

Owner of WES2 — Owner of WES1

PAGE NO. :
DATE : / /

→ Strong ES is the ES which is not dependent on any other ES for its existence.

Another Ex: Whatsapp DB



→ WES has a partial key which is unique within the key of owner identity → also called Discriminator

Ex: Emp ES can't have two dependent of same d_name if d_name is partial key of dependent.

→ PK of identifying relationship is (PK of owner, Partial key of WES)

→ PK of WES is none.

⇒ WES don't have repeated entities this is just that we can't identify them uniquely.

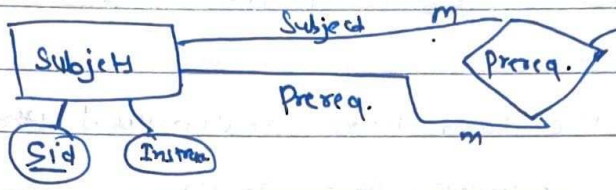
Ex: In dependents two child of 10 years can be there, but both are dlt despite being same name & age.

→ Every entity in any ES is uniquely identifiable by its own attr (SES) or using identifying relationship (WES)

PAGE NO. :
DATE : / /

→ Recursive relationships can also have diff types like:

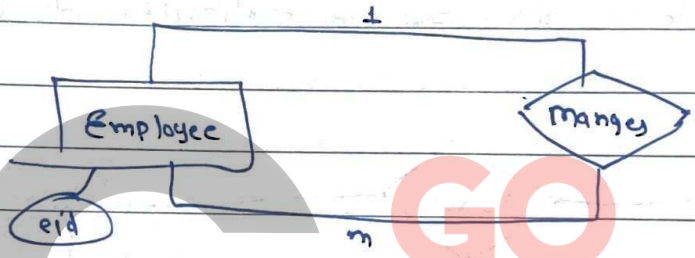
Ex. 1



One subject can have many subjects pre requisite.

Many to many Recursive Relationship.

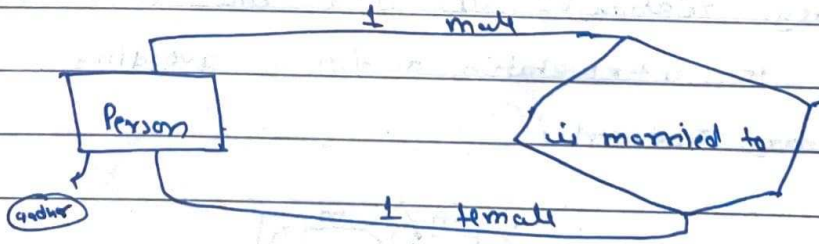
Ex: 2



One emp. manages many emp but each emp is manged by one emp only

Many to one Rec. Relationship

Ex:3



One Person (male) is married to 1 Person (female) only

One - to - one Rec. Relationship.

Trick in relationships we can see arrow as FD also like

a, b, } Not possible
a, b, }

{ Be created in M to 1 Relationship

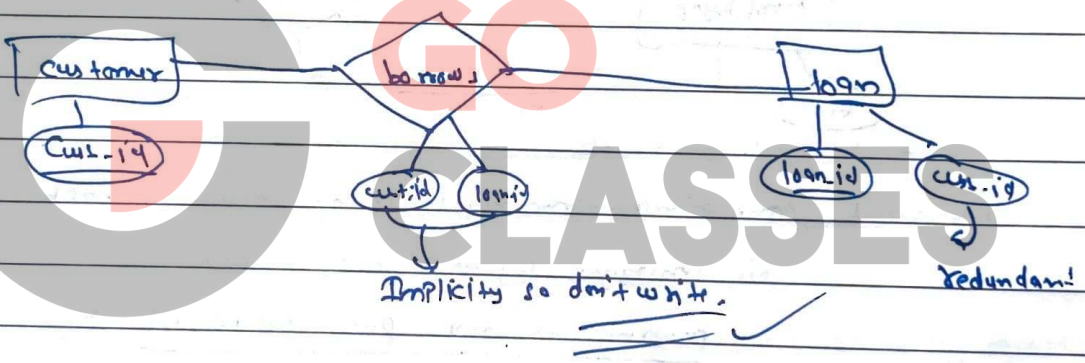
PAGE NO. :
DATE : / /

→ In ER model we can not represent FDs and foreign keys.

→ In relational mode we can't have multivalued attributes, composite attribute & weak/strong entity.

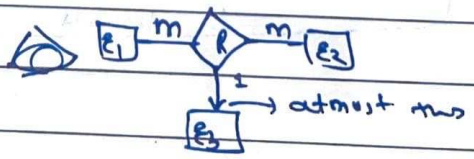
→ So do not specify one entities pk in other entity as FK, it will be redundant as association is contained by the relationship. Also don't give keys to relationship it's implicit.

Ex:



→ In non-binary relationship at most one arrow (one side) is allowed due to interpretation ambiguity avoiding

Ex: In ternary relationship



→ Now we know everything about ER model. Let's move 1 step ahead and convert it to Relational model. And then from Relational model we know to write SQL.

PAGE NO. :
DATE : / /

Imp for GATE

☆ E-R Diagrams to Relational Model:

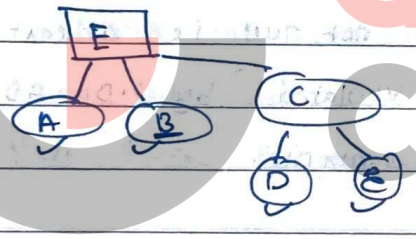
Most things are overlapping b/w the two.

There are three components of Conversion process:

a). Specify schema of Relation
b). Select PK for the Relation.
c). Specify FK references to other Relations.

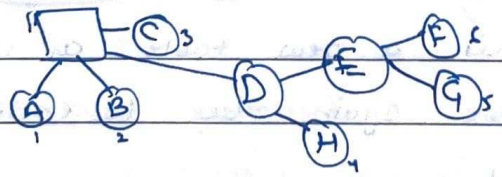
- Entity set name → Relation name
- Simple Attributes → Table columns
- Composite Attributes → Take the subparts as table columns
- Identifying Attribute → PK

Ex:



⇒ E(A, B, D, E)

Que: How many attributes in Relational model for below ERD?



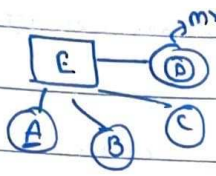
⇒ 6

→ Multivalued Attribute (MVA) are not allowed in Relational model, so in RM we create a seprate table for MVA.

→ Derived Attr → Not stored in Relation.

PAGE NO :
DATE : / /

Ex:



⇒ (A, B, C) & (A, D)
FK

So we create a whole new relation in place of MVA & use PK as FK in this new relation.

Qw: Why don't we allow in same table with many rows for MVA?
along old PK

⇒ PK will repeat and if we make MVA also PK, then it can't be null but it was null for some attributes. So, only option is to create a new relation.

Also even if it was not null i.e atleast 1 value for MVA then 2NF will be violated by P.D. AD→B & AD→C while A→C and A→D in above example.

→ MVA ⟷ 1 weak ES for each MVA → 1 new table

→ For each MVA there is a new table as we can't keep all MVA in one diff table again due to entity IC (Primary key may become null).

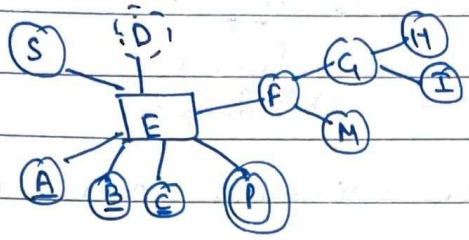
→ On delete cascade is the relievant RIC for MVA relation.

→ Complex Attributes = Multivalued + Composite Attribute.
↳ New relation with many simple attributes.

→ Derived attribute is not kept in relation, it is computed when required.

PAGE NO. :
DATE : / /

Qw: Convert ER to RM:



⇒ Two relations as:

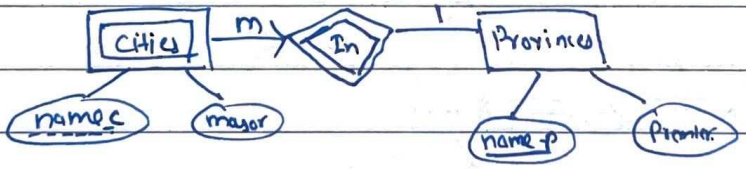
(A, B, C, S, M, H, I) ✓

(A, B, C, P) ✓ # attributes = 7+4 = 11

→ Similar to MVA, WES is also treated as new relation. Exactly same, no difference.

Here PK = PE of Owner Entity ∪ Partial key / Discriminator of WES

Ex:

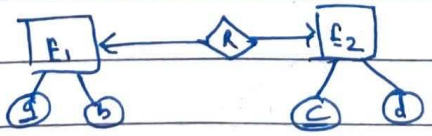


⇒ (name_p, premier)
(name_p, name_c, mayor)

} Equivalent Relations

→ For 1:1 relationship, create a new relation with PK as PK of any entity if both sides are partial.

Ex:



= R_1 (a, b), R_2 (c, d), R_3 (a, c)

↳ any one as key

PAGE NO :
DATE : / /

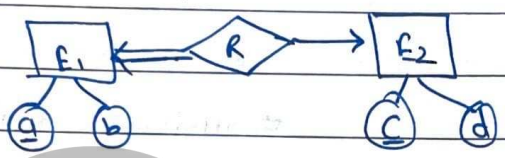
→ If our design allows null value in schema then we can also do it as

(a, b, c) (c, d) OR (a, b) (a, c, d)

a can be null as partial participation

→ For 1 to 1 mapping with total participation on E_1, merge Relationship to E_1

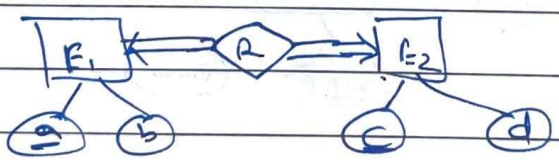
Ex.



then (a, b, c) and (c, d)

↳ It can't be null as total participation.

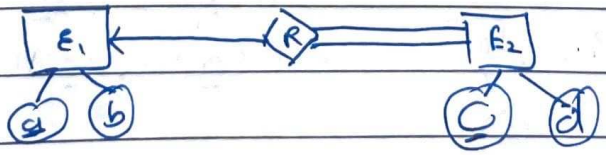
→ for 1 to 1 mapping both side total participation (bijection), create single table.



then (a, b, c, d)

↳ PK can be a or c ✓

→ for 1 to many with total participation on many side.



PAGE NO. :
DATE : / /

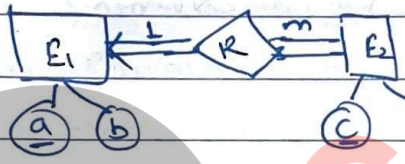
Then it will be treated as CWEs or MVA and Relations will be (a,b) and (a,c,d)

FK ↳ key will be C simply

↳ why not key?

↳ Think

→ 1 to many, total participation both sides.



Good design would be:

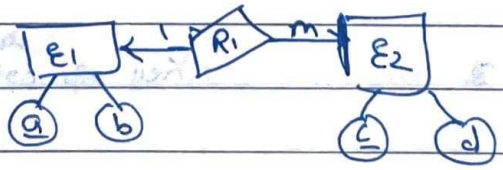
$R_1 = (a,b)$, $R_2 = (a,c,d)$

In case of single table 3NF will be violated as:

a → b

↳ not SK ↳ NPA

→ 1 to many, partial participation both sides



Create 3 diff Relations (a,b), (c,d) and (a,c)

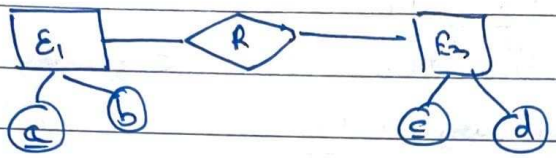
as no full participation in any ~~relation~~ side. , so creating 2 tables will have null values in it.

If null values allowed then we can do

(a,b) , (a,c,d) ✓

PAGE NO. :
DATE : / /

→ Many to Many, any participation,



fix conversion i.e. 3 Tables (a,b) (c,d) (a,c)

If we create single table then key will be ac and $a \to b$ (a subset of key), so it will not be in 2NF.

*→ Good DB design has tables with no null problem, atleast in 3NF and min. # tables. So, do conversion accordingly.

* Summary:

Cardinality	Membership	# Relations	Note
1:1	Both Total	1	all attr. in same table
1:1	Both Partial	3/2	3 if null not allowed else 2
1:1	One Total & One Partial	2	Key of optional in total one (Partial)
M:N	Both total / Both partial / One total & one partial	3	One for each entity & one for relationship
1:N	Both total	2	Key of one in many side
1:N	Both partial	3	two for entities & 1 for relation

PAGE NO.:
DATE: / /

1:N	many is total one side partial	2	key of one in many side.
1:N	many is total one side partial	3	two for entities & 1 for relationship

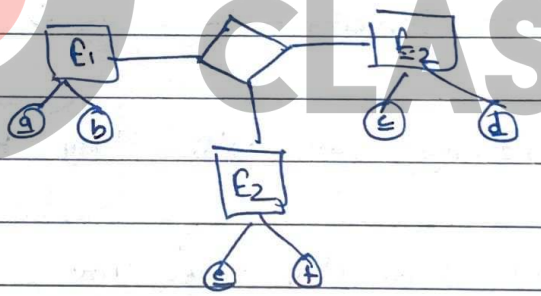
→ Do not byheart them, think, they are very logical.
→ atmost one side is one

☆ Non-binary Relationships to Relational model.

For ternary Relationships: (In gate)

Case I: All sides are many:

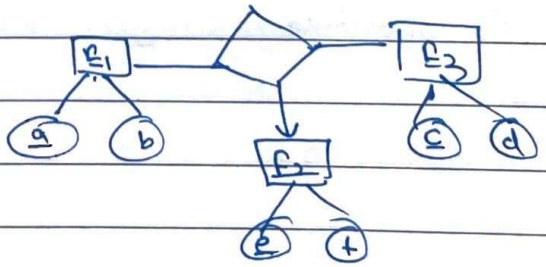
Ex:



then relations are (a,b), (c,d), (e,f) and (a,c,e)

Case II: One side is one and two side many:

Ex:



then relations will be (a,b), (c,d), (e,f) and (a,c,e)

↳ not in BK because ac → c. ✓

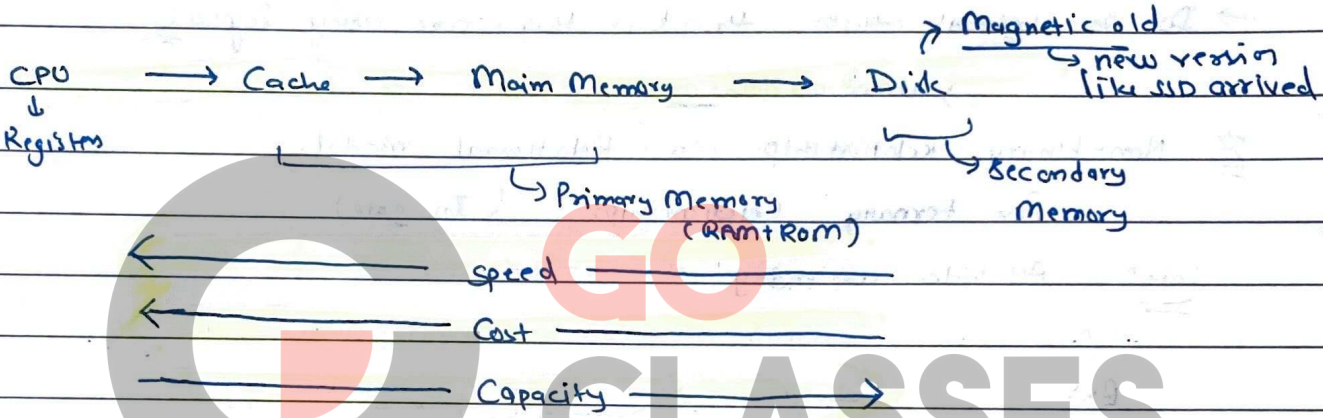
Not Related to DBMS that much

Module - 4 Magnetic Disks

PAGE NO. :
DATE : / /

Magnetic Disk:

→ We had level view of DB, but in actual vast majority of DB are stored in hard disk & fetch it in main memory for processing. Memory access is way more faster than disk access.



→ In this module, things we will study happens physically.

→ Diff areas of disk have diff access time

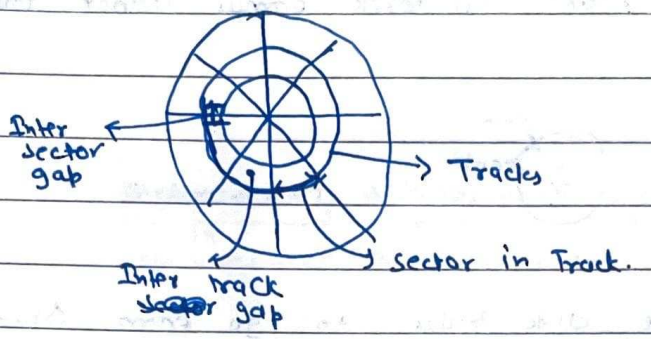
→ In main memory data is stored in form of charge & in magnetic disk bits are stored in form of magnetic poles.

→ Magnetic tape had only sequential access but in magnetic disk, Random access is possible.

PAGE NO. :
DATE : / /

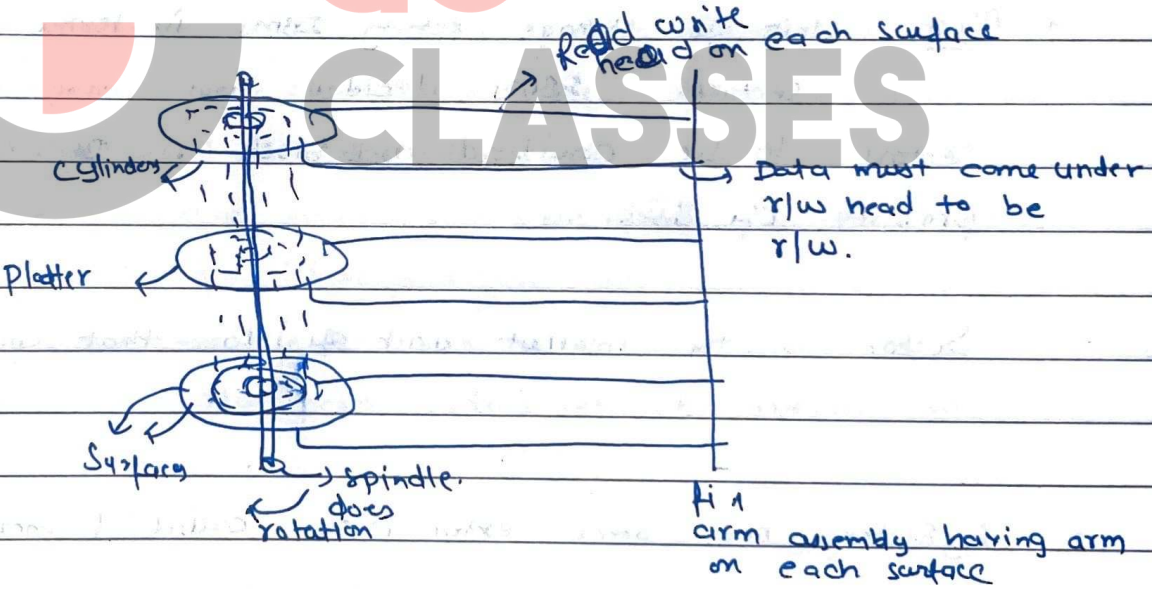
☆ Disk structure:

Particular surface:



→ #sectors in each track are equal, they just increase inter sector gap size. Also size of each sector is same.

Disk:



→ Each platter has two surfaces.

→ Virtual alignment of tracks of same diameter, on top of each other is called cylinder.

So #tracks per surface = #cylinders.

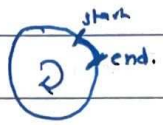
→ All r/w head move together & vertically only in same position, but only one r/w head is active at a time.

PAGE NO. :
DATE : / /

Typically sector size is 512 B.

→ **Seek time**: Time to put the r/w head on particular track.

→ **Rotational Latency**: Time taken by disk to rotate such that start of sector comes under the r/w head.



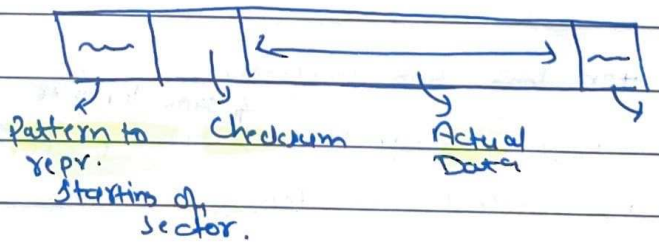
→ **Transfer Time**: Time disk takes to go from Start of sector to end of sector i.e. to read the entire sector.

→ **Block**: Unit of storage set by DBMS in terms of # sectors. DBMS decides how many consecutive sectors to be considered as 1 block. It is not a property of disk.

→ Sector is the smallest unit of info. that can be read from or written to the disk.

→ Any Sector has some extra data called formatted data.

Ex:



→ **Disk controller** interface b/w computer & actual h/w of disk. It takes actions like moving arm on getting command from high level. Checksums are also attached by disk controller.

PAGE NO. :
DATE : / /

* **Performance measures of Disks:** Main measures of the qualities of a disk are:
  - Capacity
  - Access Time
  - Data Transfer Rate
  - Reliablity.

a) **Capacity:**

#platters *2 * # tracks/surface * # sectors/track * sector size
|| || ||
surfaces * # cylinders * # track size

* **Process**

When the disk drive is operating, disk is rotating at constant speed. Now to read/write a sector, what we do is:

a) Take r/w head to desired track (Seek time)
b) Let the starting point of desired sector come under the r/w head (Rotational/Disk latency)
c) Read write the sector while entire sector passes under the r/w head (Transfer Time).

b) **Access Time:** Rotational Latency + Seek Time.
i.e. Time taken b/w request issuance to actual transfer begin.

→ If avg. latency (Rotational) is to be calculated then we consider half rotation time.

2 good ques in Google Doc
↳ Refer ✓ must

PAGE NO :
DATE : / /

→ **Data transfer rate | Throughput**: Rate at which data can be retrieved from or stored to the disk i.e. how much data (in Bytes) per second can we read | write.

UGCNET

Qu: Concern a disk with
Sector size = 512 B
2000 track | surface
50 sector | track
5 double sided platters
& avg. seek time is 10ms (Not req)

- Capacity of track (T), surface (S), disk (D) is? (in B)

⇒ T = 50 × 512 B.
S = 50 × 512 × 2000 B.
D = 50 × 512 × 2000 × 10 B.

★ How data is stored on Disks?

→ Data on a disk is first stored in the first sector of the first surface of the first cylinder. Then in the next sector and next, untill all the sectors on the first track are exhausted. Then it moves on the first sector of the second surface (remains at same cylinder i.e changes surface instead of track to save the seek time) It exhausts all available surfaces for the first cylinder in this way. After this, it moves on to repeat the process for the next cylinder.

PAGE NO. :
DATE : / /

→ Closest two records on a disk are on the same sector. In decreasing order of closeness; they can be on a) same block,
b) same track,
c) same cylinder,
d) adjacent cylinder.

It is important because closeness means time to read next record and it is surely dependent on position of data.

☆ Disk Sector Addressing:

→ By default, track no. starts from 0 and track 0 is the outermost track of disk.

→ We start from 0 because then if we are integer k then it means we have passed 'k' cylinder/tracks/sectors already.
for Ex: Cylinder 10 means we have crossed/passed 0-9 i.e. 10 cylinders already

→ There are two types of addresses:

a) Cylinder - Head - Sector (CHS): Represented as <cylinder_no, Head_no (Surface), Sector_no>

b) Logical Block Addressing (LBA): Integer no. of the sector

Ex: Assume 3 platters, → 6 surfaces, 3 cylinders & 4 sectors per track

⇒

LBA	CHS	LBA	CHS
0	<0,0,0>	7	<0,1,3>
1	<0,0,1>	8	<0,2,0>
⋮		⋮	⋮
3	<0,0,3>	24	<1,0,0>
4	<0,1,0>	25	<1,0,1>
5	<0,1,1>	⋮	⋮
		55	<2,1,3>

Start from 0 means those much crossed.

PAGE NO :
DATE : / /

* CHS → LBA

Let CHS = <c, h, s>

and #sector per cylinder = x.

#sector per track = y

then LBA = (c*x) + (h*y) + s

(c*x): sectors crossed at cylinders
(h*y): sectors crossed in above surface
s: sectors crossed in current track.

* LBA → CHS

let LBA = L

& #sector per cylinder = x

#sector per track = y

then c = int(L/x) , temp1 = L%x

h = int(temp1/y) ,

s = temp1 % y.

→ The above mentioned conversions are only if counting starts from 0, if it starts from 1 then add 1 in all.

* <c,h,s> → LBA

LBA = [(c-1)*x] + [(h-1)*y] + [(s-1)]

* LBA → <c,h,s>

c = (L/x) + 1 , h = (temp1/y) + 1 , s = (temp1 % y) + 1

If it starts from 1 then if we are on 1st it means we have crossed 1st.

→ In magnetic disk read & write are equally fast as just pass through the r/w head.

PAGE NO. :
DATE : / /

☆ Disk scheduling algorithms:

OS has many I/O devices and it maintains queue for each I/O device. for a single I/O device, there will be a number of requests I/O (read or write no issue) from various processes in a queue.

Disk scheduling algorithms deal in the way/order of executing these requests.

→ When we have sector R/W head requests from many processes then thing that matters the most is seek time for fast performance.

Now, seek time will be 0 iff all sector requests belong to the same cylinder.

Rotation speed is usually very fast then head movement so seek time matters the most.

→ So to improve the disk performance cylinder number matters for us the most. Sectors of same cylinders are not of much worry.

→ So, disk scheduling algorithm cares only about the cylinder number of the I/O request.

→ Flow is:

Disk request from Process → Disk queue → Disk scheduling algo schedules request on basis of cylinder no.

✓→ All requests of same cylinder are dealt at same time as no seek is required, so disk controller maintains a LL of requests cylinder wise.

Elevator → Lift
Escalator → Swachalit Sidiyan

PAGE NO. :
DATE : / /

Qu: If a process makes a Disk I/O request, what info must process provide to OS?

→ a) Type of opⁿ (i/p or o/p) b) Disk address of transfer.
c) Memory address (where to send) d) No. of sectors to be transferred

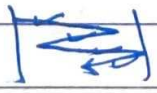
→ Disk scheduling algo
- i/p: List of cylinder no.
- o/p: order in which cylinder req. executed. Such that total seek time is reduced.

A. FCFS / FIFO: Requests are executed in the order they arrived. It is fair but not fast i.e. inefficient.

B. SSTF (Shortest Seek Time First): It is shortest seek time first. It solves problem of FCFS. service all the requests close to the current head position. It is unfair i.e some of the request might starve if requests from near keep coming. It is also not optimal.

C. SCAN (Elevator): In this disk arm start at one end and move towards other end, servicing requests as it reaches each cylinder. At the other end, direction is reversed and it comes back servicing the cylinder (like an elevator).

D. Look: It will go keep looking i.e. not go till extreme end if not any request there. It goes till last request only.



PAGE NO. :
DATE : / /

Elevator

C-SCAN: It goes from one end to another but comes back w/o servicing then again goes servicing

C-LOOK: Here also servicing is done in one direction only and head do not go till extreme end.

→ SCAN was not uniform as cylinder at ends is serviced twice very fast (that way). So this is improved in C-scan algorithm.

Ques: Which scheduling algorithm is Best?

⇒ Look > Scan.

Clook > Cscan

Everyone > FCFS

Rest are incomparable.

FCFS < (Scan, Cscan) < (Look, Clook)

→ We may have to calculate total head movements.

Avg head movement = $\frac{\text{Total head movement}}{\#\text{ requests}}$

→ If in que given disk details and sector no. (in CBA/CHS), get cylinder number and then apply algorithm.

PAGE NO. :
DATE : / /

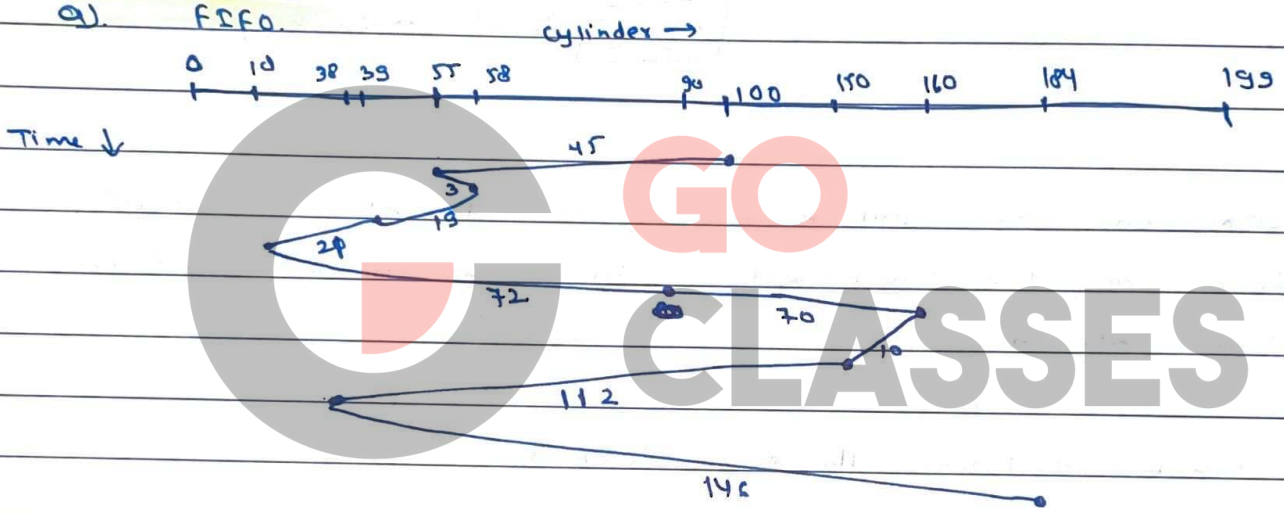
Ques: Assume disk head initially is at track 100. Disk has 200 track. Initial direction is towards increasing track no. Circular algorithm service towards only higher cylinder.

Requested traces, in the order received by the disk scheduler are:

55, 58, 39, 18, 90, 160, 150, 38, 184

Apply all algorithms. (Check all)

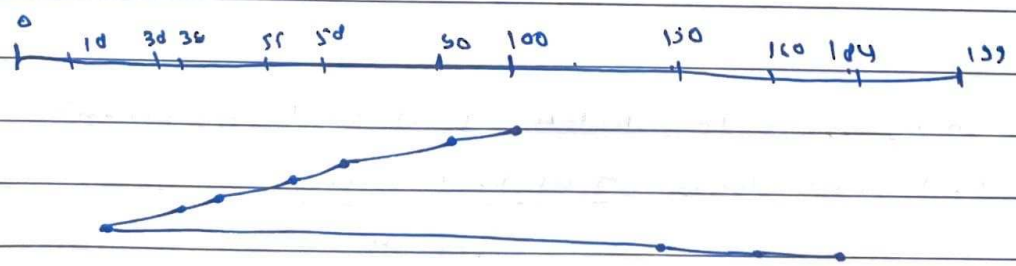
a). FIFO.



head movements = 498

Avg = $\frac{498}{5}$ = 55.3

b). SSTF

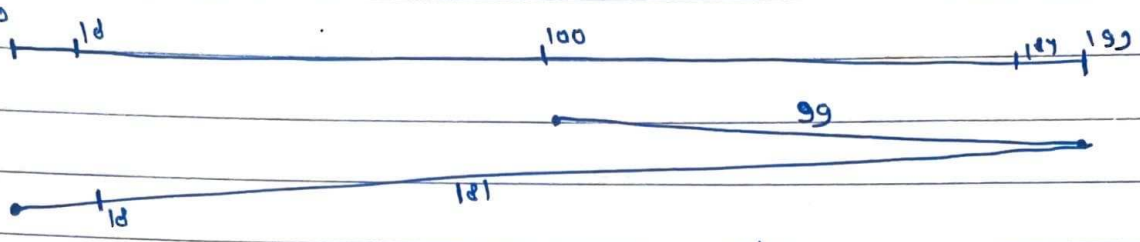


head movements = 82 + 166
= 248.

Avg = $\frac{248}{5}$ ⇒ 2[illegible].5

PAGE NO.:
DATE: / /

d) SCAN.



Serve all in middy.

head movement = 181 + 99 = 280

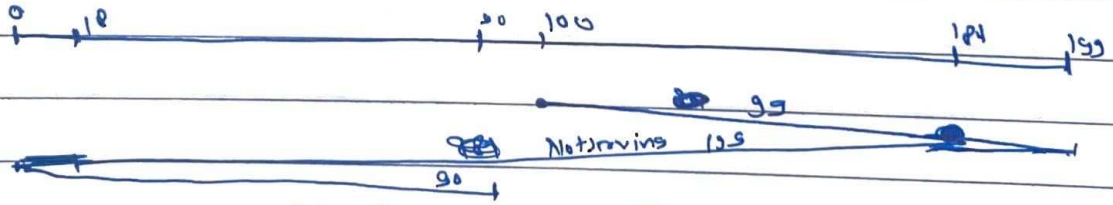
avg = $\frac{280}{9}$ = 31.11

e) Look

head movement = 84 + 166 = 250

avg = $\frac{250}{9}$ = 27.8

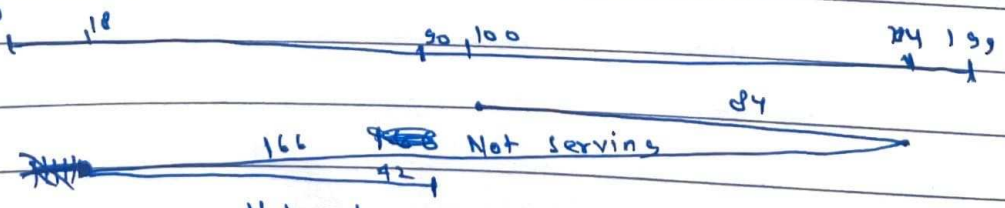
f) CSCAN:



head movement = 199 + 90 + 99 = 388

Avg = $\frac{388}{9}$ = 43.11

f) CLOOK:



head movement = 84 + 166 + 72 = 322

Avg = 322/9 = 35.8

PAGE NO. :
DATE : / /

Module-5
↳ File Organisation & Indexing

★ **File Organisation (FO)** It tells us how Database is actually stored in Hard Disk.

→ Indexing is very diff from FO, don't mix them. Indexing only simplifies our data access like index of actual book.

→ DB has many tables. There is a single file for each table / relation.

→ Each relation have many Records. Block is capable of storing 'n' records.

→ If we need any record than that block is fetched into the main memory., because block is the smallest unit of data access for DBMS. If record is changed then we also rewrite that block in our Disk.

→ There is a tradeoff in block size:
Smaller block → more transfers from disk.
larger block → space wasted due to partially filled (Internal Fragmentation)

→ Sometimes term "Page" is also referred for Block in DBMS. Block size is decided by DBMS, it's not the property of disk.

→ Blocks size ≥ Record size obviously. So a block usually contain many records.

→ Record ≡ Tuple, & Table ≡ file.

PAGE NO. :
DATE : / /

** Types of Record and Record format:

There are two types of Records:

a) Fixed length records (ie all fields have fixed size)
b) Variable length records (ie some field have variable size).

→ Varchar DT makes Record of variable length.

Ex:1 ID int, -4
Name char(30) -30
Gender char(1) -1

↑
Fixed length of size ⇒ 35 B

Ex:2 ID int -4
Name varchar(30) -0→30
Gender char(1) -1

↑
Var length record of size = 5 to 35 B

* finding field inside a Record:

→ If fixed length records then there is no issue, we already know where field inside record starts and ends.

→ There are also some record properties like "time last accessed" present inside record. These are also called Record metadata. or Record Header.

→ There are some block properties like #records in this block, these are called block header.
↳ It stores size of each field of records and all.

PAGE NO. :
DATE : / /

→ Now, we will see, how to find particular field inside a variable length record.

M-1. Use delimitor after each field

Ex:

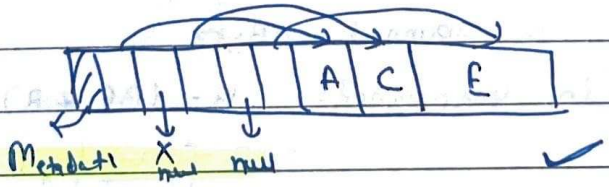
This solution is not used as we'll have to do sequential access to go to particular field.

M-2 Store address of each field in Record itself.

Ex:

In case some field is null in a Record we set its address pointer as null and don't keep anything for that.

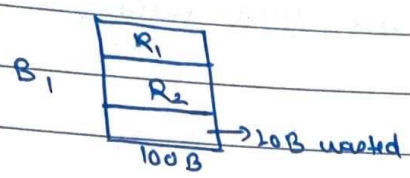
Ex: B and D are null, then

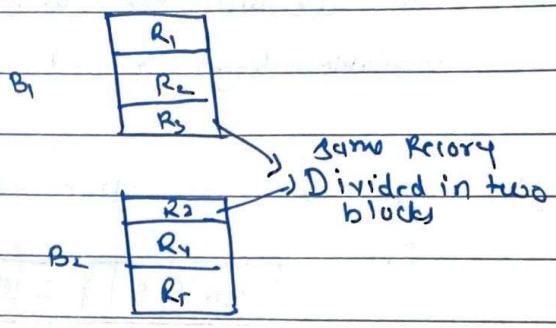


* Spanned v/s Unspanned Organisation of Records

If block size is 100 B and record size is 40 B and if we store 2 Records in block it is unspanned organisation and if we store 2½ Records it is spanned organisation.

PAGE NO. :
DATE : / /





Unspanned Organisation.
↳ Records not allowed to cross block boundaries

Spanned Organisation.
↳ Records can span in more than one block.

* Blocking factor: (BF) #Records / Block.

Ex: Block size = B Bytes, Metadata = k B, Record size = R B then

$(BF)_{unspanned} = \left\lfloor \frac{B-k}{R} \right\rfloor$ and $(BF)_{spanned} = \frac{B-k}{R}$

Space wasted in spanned = 0 Bytes

Space wasted in unspanned: $B - (BF * R)$ Bytes

$= B - \left[\left\lfloor \frac{B-k}{R} \right\rfloor * R\right]$ Bytes

→ With fixed length record, unspanned organisation is used mostly and with var. length record any can be used. (So that we know field addresses)

PAGE NO. :
DATE : / /

NOTE: By default, fixed size record and unspanned organisation are taken.

→ for var. length record, each blocks can have diff no. of records, so Blocking factor in that case is avg. no. of records / block.

Que: Var. length r records and BF is b, then # blocks req. for spanned organisation?

⇒ $\left\lceil \frac{r}{b} \right\rceil$ blocks ✓

Que: Let avg. record size for var. length records be R and # records are r. Block size is B Bytes & Block header is k Bytes. Then, using spanned organization, # blocks?

⇒ $\left\lceil \frac{R * r}{B-k} \right\rceil$ blocks ✓ (Total space; $B-k$: Block size usable)

⇒ $\left\lceil \frac{r}{BF} \right\rceil$ as $BF = \frac{B-k}{R}$ ✓

☆ **Organization of Records in File:** Now we are having set of records, we have to determine how to physically place them on disk, in what order.

Conceptually, order of records does not matter, but at physical level it matters. There are two ways to organise records in file:

PAGE NO. :
DATE : / /

A. **Heap File Organisation:** (Not min heap/max heap, it's like heap of clothes)
Here, any record can be placed anywhere in the file. There is no ordering of records. Simple & basic organisation. New records are inserted at EOF.

→ Insertion is Efficient, but searching is costly.
↳ b/2 blocks searched at an avg for each record

B. **Sequential File Organisation:** Records are stored in sequential order, according to the value of a "search key" of each record.
↳ Ordering field.

→ Insertion is costly because we have to keep it ordered

→ Searching becomes efficient as we can do binary search.

→ Binary search can be done on block rather than on the records. Binary search can only be used when searched on ordering field. From other field's pov, it is just heap organisation.

→ Ordered files are stored on contiguous cylinders to minimise the disk time.

PAGE NO. :
DATE : / /

* **Block Address:** Address of the first sector (cylinder or LBA) of that block.

* **Record Address:** A record can be identified by giving its block address and the offset of the first byte of the record within the block. (i.e. address within block)

Ques: Suppose record has following fields: 15B, 2B, 10B, 8B.
How many byte record takes if:

a) field can start anywhere.
→ 15+2+10+8 ⇒ 35

b) Field must start at a byte multiple of 4.
→ (Remember structure alignment in C)

$(15+1) + (2+2) + (10+2) + (8+0) = 40\ B$

Pad

Ques: In google doc ✓ (Good que)

* **Disk space management:** In this, we will look into two major issues involved in managing disks.

A **Block size:** We know file is split into no. of contiguous fixed blocks that need not be adjacent.

→ Now, having a large block size means that every file ties up an entire cylinder (even if file size is 1 bit).

PAGE NO. :
DATE : / /

It means that small files waste a large amount of disk space.

On the other hand, small block size means that most files will span multiple blocks and thus will need multiple seeks to read them, hence reducing performance due to increased time.

* So, if the block is too large, we waste space, if it is too small, we waste time, it's a tradeoff.

Que: Consider a disk with track size 1MB, rotation time of 8.33 msec and an avg. seek time of 5ms. Time to read, 1 block of 1KB in ms is?

⇒ Avg. rotational latency = $\frac{8.33}{2}$ ms

So, total delay before actual transfer = $\frac{8.33}{2} + 5$ ms.

After this we'll transfer at 1MB in 8.33 ms.

So block transfer time = $\frac{8.33}{2} + 5 + \frac{8.33}{2^{20}}$ ms

Problem-2

B. Keeping track of free blocks: Since the space is limited, we need to reuse the space from deleted files for new files.

To keep track of free disk space, O maintain a free space list

PAGE NO. :
DATE : / /

There are two methods to keep track of free blocks:

I. **Bitmap / Bitvector:** We have a vector where each bit repr. a block. If bit is 1 block is free and bit is 0 repr. allocated block.

Ex: Consider a disk with blocks 2, 3, 5, 7, 9, 10, 11, 15, 16 are free and rest are allocated then bit vector will be:

0011010101110001110000---

For this method, disk with n blocks need n bit vector which is stored in main memory mostly.

Advantages of this approach are:

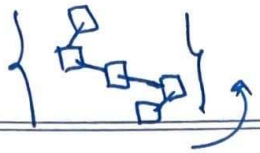
- Its relative simplicity.
- Its efficiency in finding the first n consec. free blocks.

Ex → A 1.3 GB disk with 512 B block will need bitmap of size 332 KB to keep track of free blocks.

II **Linked List / Freelist:** This method consists of using a linked list of disk blocks with each block holding as many free disk pointers as fit in it.

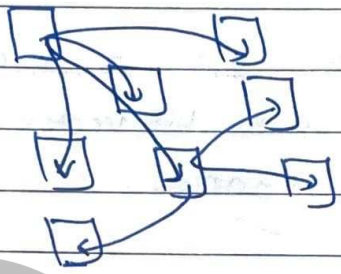
→ Store the addresses of n free blocks in the first free block. The first n-1 blocks are actually free, n^{th} block contains the address of another n free blocks and so on.

PAGE NO. :
DATE : / /



(If we had direct LL of one to second and second pain to 3rd, then it would be inefficient as many block accesses were needed to know diff free blocks)

This method looks like:



All free blocks.

→ (Not eating useful space)

NOTE ↔ There is NO space wasted in this method as, when block will be used pointer will be removed. So zero space overhead. Bitmap had fixed space overhead.

** What happens when we create file & delete file?

⇒ To create a file, we search the free space list for the required amount of space and allocate that space to the new file. This space is then removed from free space list.
When a file is deleted, its disk space is added to the free space list.

PAGE NO. :
DATE : / /

Std. Gate CS 98

Ques: Q) Disk address require D bits. For a disk with B blocks, F of which are free, state the condition under which the free list uses less space than the bitmap.

⇒ We want

(D*F) bits < B bits

$\Rightarrow \frac{F}{B} < \frac{1}{D}$

(F/B = a % of free blocks.)

So, fraction of free blocks should be less than $\frac{1}{D}$

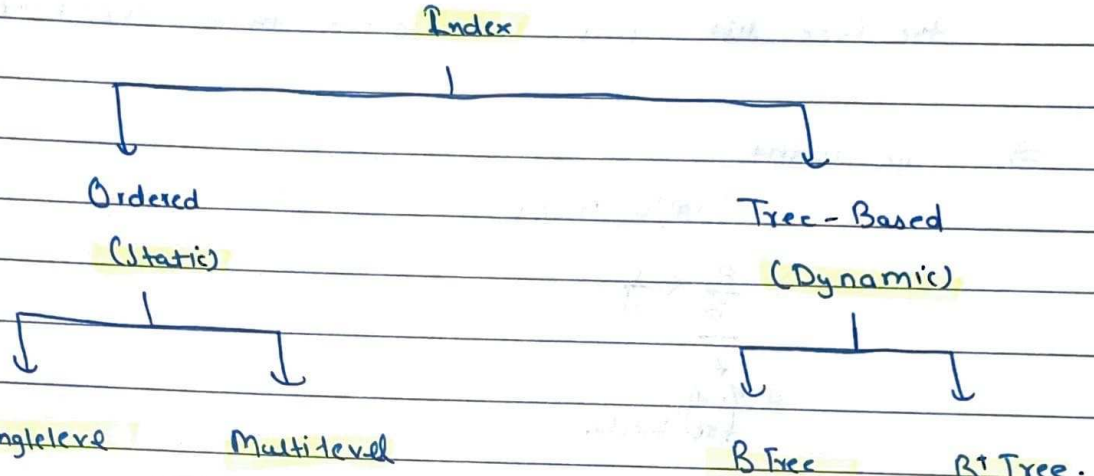
Ex: For 16 bit disk address

$(6\%) = \frac{100}{16}\%$ of blocks can be free atmost for free list less than bitmap.

Indexing

PAGE NO. :
DATE : / /

Classification:



→ In disk, we already have our data in some blocks.

→ Index are extra file that are useful and based on our data, but they do not modify the data.

Analogy to Book

Book + Index → Topic name
Data. ↳ Index → address of topic (i.e. pg. no)

↳ Index

Search key — Block Pointer

So any entry of index is <key-value, address of record>.

→ Indexes can be on diff attributes of a Relation.

Ex: R(A, B, C, D)

↳ Actual data in disk

we can have upto 4 indices on R

↳ in Disk.

PAGE NO. :
DATE : / /

→ Search key ≡ Index field.

→ To gain fast access (Random) to records in a file give value of an attribute, we use index of that attribute.

→ Indexes can themselves be sorted (we will see now), or they can be hashed also (seen in DS).
↳ i.e. unsorted.

★★ Single Level Ordered Indices:

Here index is sorted for sure, and, file may or may not be sorted.

→ In an ordered index file, index entries are stored sorted by search key value.

→ Data file on which we are creating index can be heap file (unordered) or sequential file (ordered by ⊥ field).

→ Indexed file i.e. Data file need not be sorted for creating index file.

→ There are three types of single level ordered indexices:

a. Primary Index
b. Clustering Index
c. Secondary Index.

PAGE NO. :
DATE : / /

A. Primary Index:

If data file is sorted on a key attribute 'A', then index on 'A' is called Primary Index.

→ Requirement:
- Sequential data file sorted on A
- A is CK.

then Index on A is primary index.

→ Our book's index is not primary index bcz, our book is not sorted by topic name i.e. key of index.

→ We can't create primary index on a Heap file. Only possible on sequential file.

→ Ordering field is field on which physical file is ordered, so now if ordering field is also a key field then index on that field is primary index.

→ At most 1 primary index at a time is possible because file can be sorted (stored) on 1 field only at a time.

* Implementation of Primary Index:

There are two types of implementation possible:

a). Dense Implementation: When we have 1 entry in index file for each search-key value.

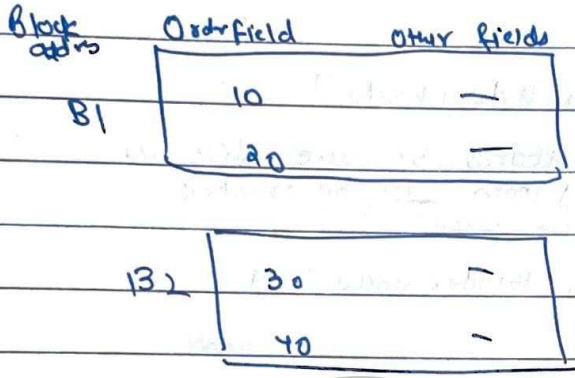
b) Sparse Implementation: When we have entry for not all record instead of each record. block or anything else

#index entries < #records

PAGE NO. :
DATE : / /

→ Mostly we store block address, bcz, record is easy to find once we access the block. Accessing block is costly.

fx: let., records be



Then Dense implementation will be:

10	20	30	40	50	60	70	80
B1	B1	B2	B2	B3	B3	B4	B4

and sparse imp. will be.

10	30	50	70
B1	B2	B3	B4

→ first record of block is known as block anchor.

→ In case of primary index, it makes sense to have sparse implementation only.

→ If we have I blocks for Index, then we can do binary search on index and then fetch the particular block.

block accesses = $\lceil \log_2 I \rceil + 1$

($\lceil \log_2 I \rceil$: Index blocks; $+1$: for data block)

→ Primary Index uses Sparse Implementation / Index.

PAGE NO. :
DATE : / /

→ In sparse implementation, in index entry, we can not have record pointer, we just have the block pointer.

→ # index entries in primary index = # Data Blocks.

→ Cost w/o primary index = $\lceil \log_2 \#\text{data blocks} \rceil$

↳ we can do binary search because file is sorted as primary index can be created

Cost with primary index = $\lceil \log_2 \#\text{index blocks} \rceil + 1$

and in general #index block <<< # Data blocks.

Good ✓

Ex: Given, ~~# records = 69000~~

data blocks, b = 9500 blocks

Block size, B = 4 KB = 4×2^{10} B

(OKF) Ordering key field length, V = 15 B

Block pointer length, P = 6 B

a) No. of records in Index file = b = 9500

b) Size of entry in Index file = P + V = 15 + 6 = 21 B

c) Blocking factor = $\left\lfloor \frac{B}{21} \right\rfloor \Rightarrow \frac{4 \times 2^{10}}{21}$ records/block = 195

d) No. of index blocks = $\frac{9500}{BF} = 49$ blocks

e) Max. block accesses w/o Primary Index on OKF = $\lceil \log_2 9500 \rceil$ → bcz ordered

= 14

f) Max block accesses with Primary Index on OKF =

$\lceil \log_2 49 \rceil + 1$

= 6 + 1 = 7 ✓

g) Max. block accessed w/o Primary Index on non-ordering field

= 9500 ✓

PAGE NO. :
DATE : / /

B. Clustering Index: If the file containing the records is sequentially ordered by a non-key field A, then Index on A is clustering index. This Ordered Nonkey field is called clustering field.

for C.I:

- File must be ordered on field f
- f must be non-key (if key then PI).

→ Book's index is not ~~also~~ CI also, nor PI.

→ In heap file we can't have CI.

→ At most one CI can be there on any data file.

→ If CI then no PI, if PI then no CI

→ #PIs + #CIs can not be more than one.

* **Implementation:** Here we have index entry for each distinct value of clustering ~~index~~ field

#index entries = ~~#~~ clusters = #distinct values of clustering field.

→ Index entry have block pointer of first record that has record of that field value as many records can have same field value (as non-key).

→ It's Implementation is also sparse as #index entries are less than #records.

PAGE NO. :
DATE : / /

Ex: Let data be

# Block	Ordering field (Non-key)	other fields
B1	10	
	10	
B2	10	
	20	
B3	30	
	30	
B4	30	
	30	
B5	40	
	50	

then, index file will be

10	20	30	40	50
B1	B2	B3	B5	B5

Ex: Let us have an ordered file
with #records = 300,000
Block size, B = 4096 B
Ordered by attr. Z and there are 1000 distinct Z's in file.
assuming even distribution of Z in file.
Z field length = 5 B
Block pointer = 6 B
Record size = 100 B
If we created index on Z then

a) #Index entries
= 1000

PAGE NO. :
DATE : / /

b) Blocking factor of Index

$\Rightarrow \left\lfloor \frac{4096}{5+6} \right\rfloor = 372$

c) # index blocks

$= \left\lceil \frac{1000}{372} \right\rceil = 3.$

d) max # disk block accesses w/o Index for 2 value

$\Rightarrow \lceil \log_2 7500 \rceil + x$ as 7500 data blocks

$= 13 + x$

x is we may have to search

x more block if it is last record with 2 values.

$x = 8$ as 2 is uniform, avg. 300 records of same 2 value

So total $= 13 + 8 = 21$

1 inc. in binary search

$\frac{300000}{40} \pm 7500$

worst case

e) max # DBA with Index on 2

$\Rightarrow \lceil \log_2 37 \rceil + 9$

$= 11$

f) If index already in MM:

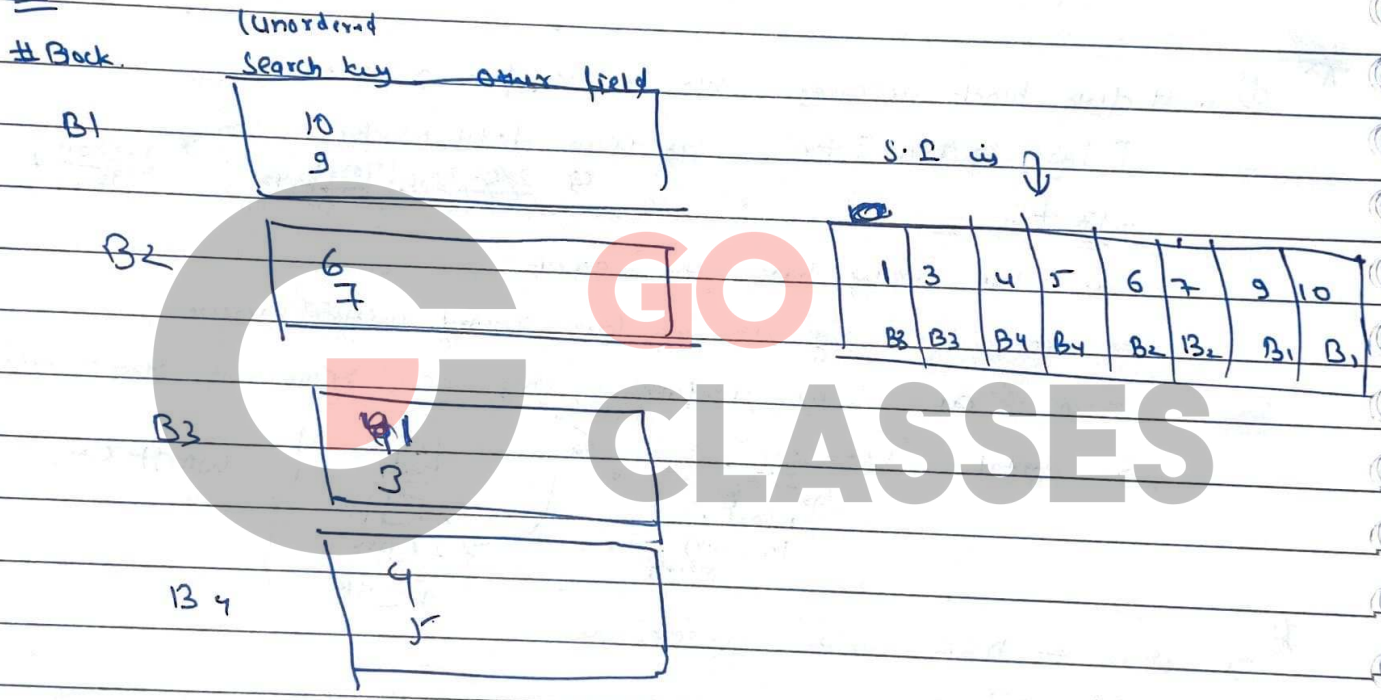
$\Rightarrow$ 9 Disk blocks in worst case.

PAGE NO. :
DATE : / /

C Secondary Index: It is an index on non-ordering field. (key or non-key).

→ Here we need to have dense index implementation.
→ We have record pointer in dense imp. by default. (both possible but)

Ex:



If SI is on key field then we just simply keep 1 entry for that record.

But if SI is on non-key field then we have imp:

a) Var length index record

key	1	2	3	4	7
Block ptr	B1	B2	B3, B7	B6	B7

} Not Preferred

PAGE NO. :
DATE : / /

b). Keep many entries for same key

Ex: Search key 1 2 3 3 4 5 5 } Not Preffered
Block Ptr B2 B3 B1 B7 B6 B1 B2

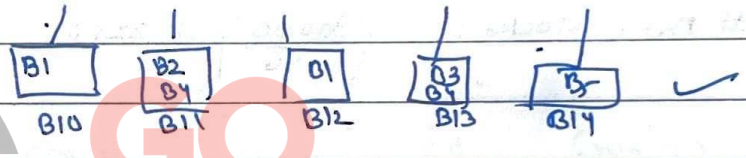
Reason in multilevel index

A new level of pointers ↓

c) Have diff block for each entry (default & preffered imp)

Ex: Search key 1 2 3 4 5

Block



i.e. each search key points to a block which holds block pointers of Data file.

→ Book's index is secondary index.

→ SI can be created on Heap file also.

→ More than 1 SI is also possible.

→ If Index field is key
#Entries = # Records
If index field is non-key
#Entries = # distinct value of field
} Dense Implementation

PAGE NO. :
DATE : / /

Ex: # data file records = 90,000
Record length = 100 B
NOF length = 15 B
Block size = 4096 B
Record pointer length = 7 B

Index on key NOF (non-ordering field)

a) BF of data file = $\left\lfloor \frac{4096}{100} \right\rfloor$ = 40 R/Block

b) # Data blocks = $\left\lceil \frac{90000}{40} \right\rceil$ = 2250

c) Index Entry size = 15+7 = 22 B

d) BF of Index block = $\left\lfloor \frac{4096}{22} \right\rfloor$ = 186 R/block.

e) # Index entries = 90,000

f) # index blocks = $\left\lceil \frac{90000}{186} \right\rceil$ = 484 Index blocks

g) Max Access cost using BS :
$\lceil \log_2 484 \rceil + 1 = 10$

h) max. Access cost w/o IS :
2250 ⇒ Can't apply binary search.

Avg : 1125 blocks ✓

PAGE NO. :
DATE : 30/10/2023

30 questions on Single Level Index (Ordered). (Solved)

Summary:

Index:

- enables efficient retrieval of a record. given values of certain attributed i.e. indexing attributes.

Primary Index:

- Index on Ordering key field (Sparse)

Clustering Index:

- Index on ordering non-key field. (Sparse)

Secondary Index:

- Index on any non-ordering field. (Sparse if on non-key, Dense usually)

NOTE Index is not always helpful

Ex: Heap org + Secondary Index and we do select *, then indexing is worse than file scan.

In range queries SI is burden.

PAGE NO. :
DATE : / /

Ordered Multilevel Indexes:

So far we have seen single level ordered indices.

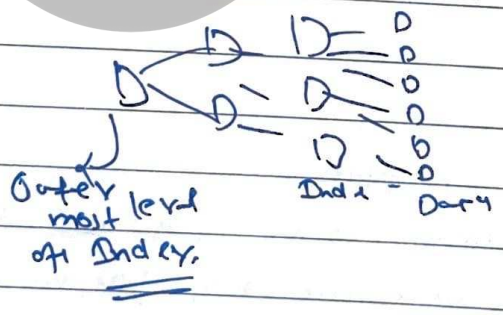
Here I/O cost = $\lceil \log_2 I \rceil + 1$

- $\lceil \log_2 I \rceil$ → To get index block
- $+1$ → To get data block.

We can further reduce this I/O cost by:

a) We put index in main memory, but it can be huge & MM cannot accomodate indexes of huge size as it has many other things to accomodate.

b) Multilevel Index i.e. Create Index of Index files. Keep doing them and at last you are left with 1 block. Now store that outermost level index block in MM.



If we choose Multilevel Index (MI) then we go till we don't have just one block of Index. at outermost level.

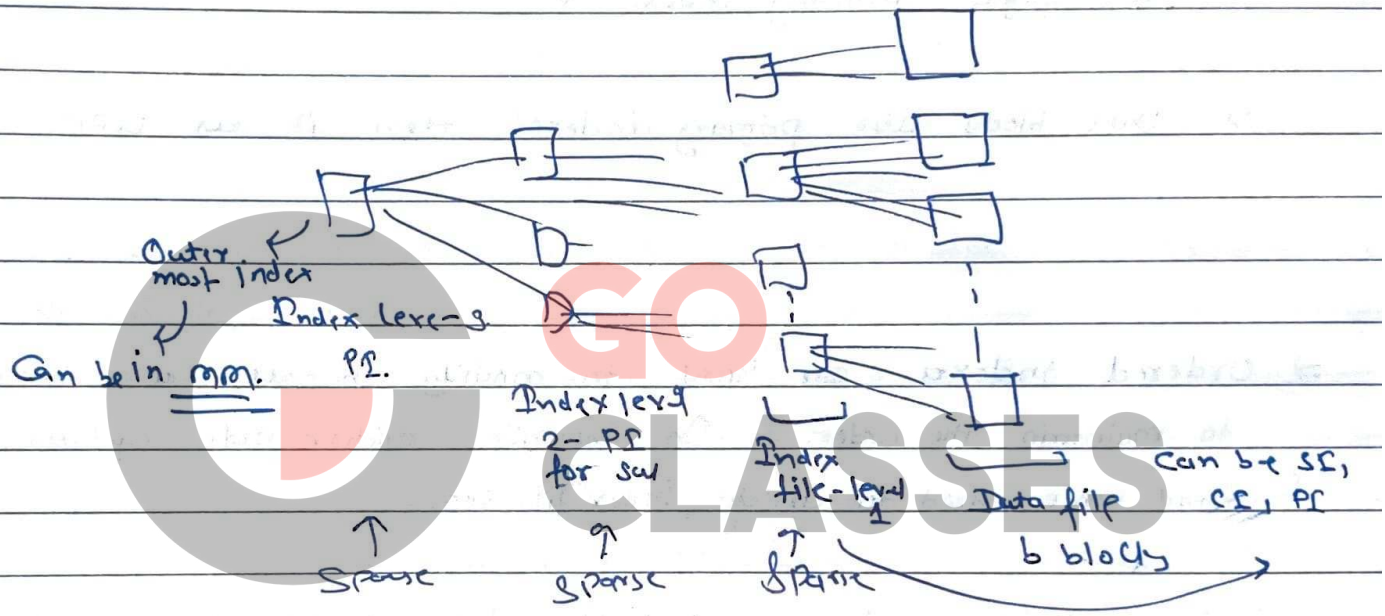
→ Multilevel xI means xI on data

PAGE NO. :
DATE : / /

~~Now indexes~~ :

** Ordered Multi level Indexes:

In the inner level indexing can be PI, CI or SI. Now whatever the type of indexing it was, on outer levels, it is always PI as it was sorted as well as key.



Ques: Assume 1st level is PI on datafile & 2nd level is dense, then what's the problem?

→ The same index file keep copying at all index levels and we will never reach a blocked index level. So, outer index levels must be sparse.

~~Note~~: # ~~memory~~ Disk block accesses = levels + 1.

→ MLI is better choice as compared to binary search on single level.

(Que in Doc)

PAGE NO. :
DATE : / /

☆ Indexed Sequential Access Method (ISAM):

→ It is just multilevel PI + Sequential file ordered by key.

→ It was used in early IBM systems.

→ Har jagah primary index. ✓

→ Data blocks also primary indexed then it is ISAM.

⇒ Ordered indexes are hard to modify because we have to maintain the order. On insertion anchor node updates and hence need to update index file too.

→ In multilevel index modification becomes even more harder So, we prefer these indices when modifications are rare and that's why these indices we have seen are known as Static Index.

→ For dynamically changing data files we need modification friendly indices say dynamic indices.

- B+ Tree Index
- B Tree Index

PAGE NO. :
DATE : / /

B & B⁺ Tree Indexes.

Tree structures we have studied in DSA like BST, AVL, Heap, etc. are in-memory DS, they are not suitable for Disk. Because, in mm a node was a structure having few bytes only, so, suitable for memory as we access memory in words, but, we access disk in block, and giving one block for few bytes node will make it inefficient. (Lot of space/block waste)

That's why we need maximum sized nodes in Disk DS. So that block can be completely utilized.

Core idea of disk DS is to put a lot of keys in a single block.

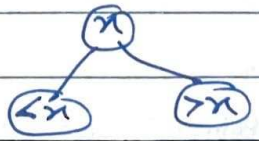
Our Convention

Level-1
Level-2

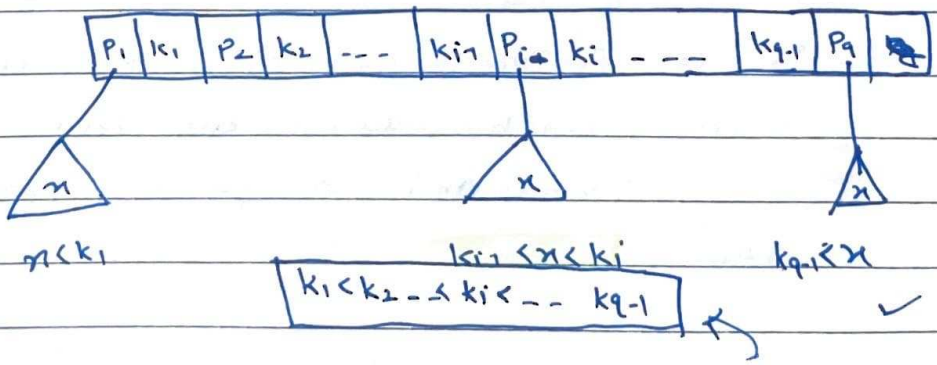
Height=1 (in terms of edges)

⇒ Obvious Rules of any search tree: (BST, B, B⁺, AVL, ~~Heap~~)

a) Left of node is less than node and right is greater



Or.



b) Keys within a node is always sorted.

PAGE NO. :
DATE : / /

In a node, k keys splits the key space into k+1 pieces

We have two main motivation while designing the in-disk datastructure: ↓ (bcz node is a block)

- Minimizing the number of levels in tree to reduce i/o access (Cost.)
- Block space utilized as much as possible i.e. nodes must be as full as possible.

→ B-Tree addresses both of these problems by specifying additional constraint on the search tree.

★ B-Tree

NOTE ↔ There is a parameter associated with each B-tree index, called Order of B-Tree, whose definition might differ and is always given in question. By default,

** Order is maximum no. of tree pointers (Block/Node) per node **

Here Block ≡ Node

& B.P ≡ Node Pointer ≡ Tree Pointer ≡ Child Pointer.

→ In a B Tree of order p, we have obvious search rules, all leves must be on same level and a special space utilization rule.

PAGE NO. :
DATE : / /

* **Space Utilization Rule:** (≥ 50% utilization) if P is order; then [OR Child Nodes]

Root Node : 2 BP to P BP
Non Root Node : $\lceil \frac{P}{2} \rceil$ to P BP
↳ i.e. ≥ 50% ✓

} Minimum req. of a node.

* **B-Tree Index Node structure:**

BP_1	<key1, record P>	BP_2	<K_2 RP>	- - - -	BP_i	<K_i RP	BP_{i+1}

i+1 = P ✓

record pointer points to where the key is, and BP points to the tree pointer.

NOTE ↔ Structure of leaf and non-leaf is same, non-leaf just sets BP to null but structure is same.

→ # max. keys / node = p-1 ✓

→ If node has x keys then x+1 Block pointers.

So, space utilization in terms of keys:

Root node : 1 key to P-1 keys
Non-Root Node : $\lceil \frac{P}{2} \rceil - 1$ to P-1 keys

} P is order.

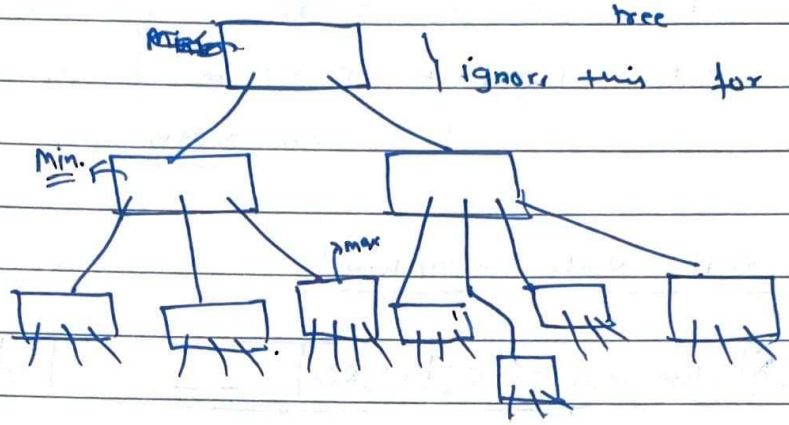
→ Given a tree, if we are asked possible orders, it can be atleast max. #children in any node of tree and atmost 2* min. #children in any node such that space utilization rule is fulfilled. [other than root]

PAGE NO. :
DATE : / /

for ex: For following B Tree possible order are 4, 5 and 6.

max. # BP. in node of tree

2* min. # BP in node of tree



So, if order is p then

Root → 2 BP to p BP

Non-Root → $\lceil \frac{p}{2} \rceil$ BP to p BP

node ✓ #key(node) i $\lceil \frac{p}{2} \rceil - 1$ ≤ i ≤ p-1 ← This should be satisfied.

→ Minimum order of any search tree is 2, if it is 1 then search tree will simply become linked list.

→ Record Pointer (or Data pointer) can be BP or PR (BP+offset) anything, that's not our problem. We have concern about child block pointers.

Ques: Block size = 4096 B, key size = 4B, BP = 8B, if no header info on block then order of B tree is ?
(Order is max. tree pointers).

⇒ Let order = p. p·P, (p-1) key

$P(8) + (P-1)(8+4) = 4096$

$8P + 8P + 4P - 12 = 4096$

$20P = 4108 \Rightarrow P = \lfloor 4108 \rfloor$

$= 205$ Order

PAGE NO. :
DATE : / /

* Searching in B tree :

Obvious algorithm, search in the node till you find a greater key and then go to prev. BP, do same till you reach leaf or find key. Finally we have data pointer with key, stored.

I/O cost in searching is # nodes we checked. It can be atmost $\lceil \log_{p/2} keys \rceil$ because of these much levels where p is order of node. In worst case each node is half filled.

T.C. of search in binary tree. (worst case)

time to search in node = $\log_2 (p/2)$
↳ binary search on $p/2$ keys of node.

nodes in worst case = $\log_{p/2} keys$

Total T.C. = $[\log_2 (p/2) * \log_{p/2} (keys)]$

$= O[\log_2 p/2 * \log_{p/2} (keys)]$

p is constant ↳ base don't matter

So T.C. : $O[\log_2 (keys)]$

Also, $\log_x y = \frac{\log_a y}{\log_a x}$

So $\frac{\log_2 p/2 * \log_2 keys}{\log_2 p/2}$

$= O(\log_2 keys)$

OR

PAGE NO. :
DATE : / /

→ With n keys AVL and B tree both are taking $\log_2 n$ time because in B tree we have less levels but we have to also search within node.

* T.C $\neq \log_2 n$ its $\log_2 n$ for B Tree. ✓

* **Insertion in B-Tree:** Insertion in B Tree starts at leaf node always.

In this algo we take a key, find a leaf node where it belongs, put it there. Now if order is maintained then good.

If insertion makes node out of order we split that leaf node in two parts and send middle node to parent node and ~~do~~ divide that leaf node into two children.

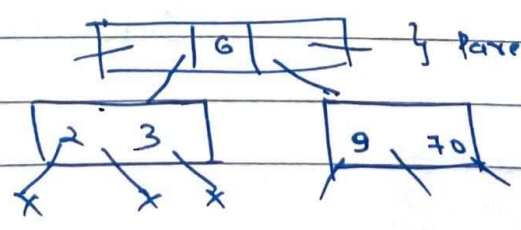
If parent node is also out of order we repeat the process till root. Root could also split in worst case.

Ex: Leaf node: [2 | 6 | 7 | 9] (nw, nw, null, nw) Order = 5, keys = 4

Insert 3 ↓

Leaf node: [2 | 3 | 6 | 7 | 9] } full now

insert 9 ↓



} Parent node

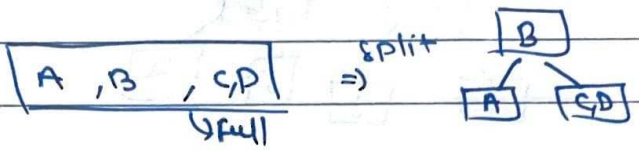
} leaf splitted in two parts.

PAGE NO. :
DATE : / /

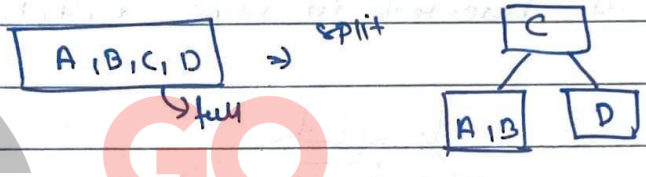
→ If order is odd then #key max can be even and on split we take the middle node to parent.

→ If order is even then we can be :

a) Right biased : More to right child.



b) Left biased : More to left child



NOTE: Always stick to same biasing for whole tree consistently else it will be wrong (Right preffered). ↳ if not in qu.

DEY:
Ex: Practice Creating B Trees using keys : ↓

21, 34, 65, 78, 90, 09, 23, 11, 55, 76, 22, 23

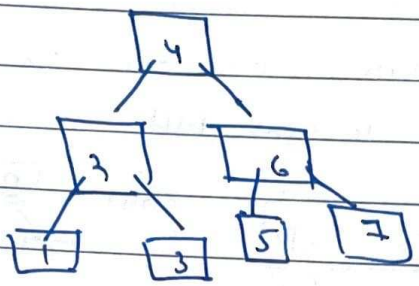
a) Order = 4 , Left Biased
b) Order = 4 , Right biased
c) Order = 5.

NOTE: Sequence/ ~~Order~~ in which keys are inserted are of very much importance, we can even get diff B Trees of diff heigh if we insert keys in diff sequence.

PAGE NO. :
DATE : / /

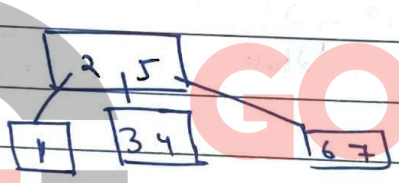
Ex: Order = 3 keys = {1, 2, 3, 4, 5, 6, 7}

Tree is inserted in ~~asc~~ seq. 1, 2, 3, 4, 5, 6, 7



Order = 3
#keys max = 2 ✓

Tree is inserted in seq: 2, 4, 1, 6, 5, 3, 7



Same Order = 3
#max keys = 2 ✓

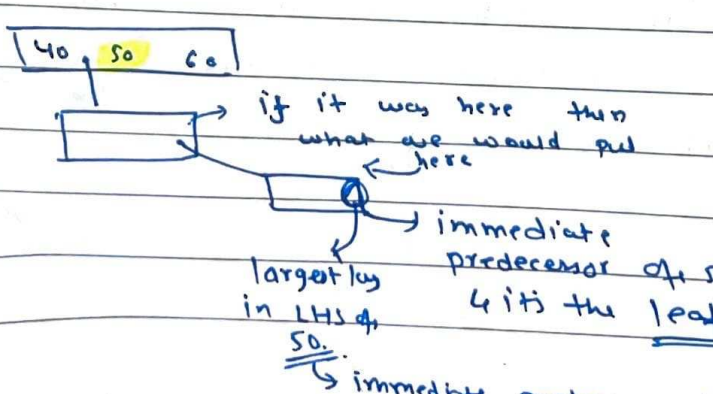
→ Insertion in B Tree can propagate split from leaf node to the root node.

→ If B tree has only one node then that node is leaf node as all BP are null.

→ In B Tree, all leaves are at the same level.

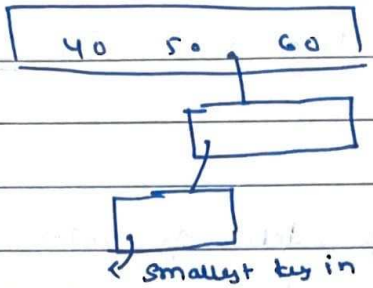
→ Immediate predecessor and immediate successor of every key of non-leaf is always in a leaf.

Ex:



PAGE NO. :
DATE : / /

Similarly successor



i.e. immediate successor of 50.

* Immediate successor / predecessor of leaf can be in leaf or non leaf.

For Ex.:

* Relationship b/w No. of leaf nodes & no. of keys in non leaf nodes:

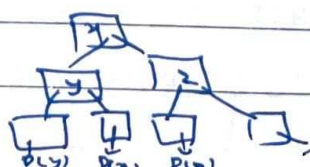
> leaf nodes = 1 + keys in non-leaf nodes.

This can be ~~proove~~ understood by predecessor & successor thing. Each key of non-leaf node has a predecessor in unique leaf node so leaf node ≥ keys in non-leaf. (Left / Right-most)

Now the ~~last~~ leaf node only do not have any key's predecessor, so,

leaf nodes = 1 + keys in non leaf nodes

(Same thing can be understood in terms of successor also)



PAGE NO. :
DATE : / /

→ In B tree, if search key is distinct set of keys then no key will repeat in two nodes in B Tree.

NOTE ↔ If in que, diff def. of order is given then, somehow get back to our def.

Ex: If order is no. of keys then:

(p)(key+RP) + (p+1)(BP) ≤ Block size. ✓

(max # BP)(BP) + (max # BP − 1)(key+RP) ≤ Block size.

↳ Now, give any any definition.

NOTE ↔ Do not forget that order of leaf and non-leaf node will be same bcz we have to reserve the space BP even though they are null in leaf nodes.

Que: Order p of B Tree is defined as: "Each b-tree node except root can have atleast p and at most 2p block pointers. Root can have at least 2 & atmost 2p tree pointers (Def. by Cormin)

Given block size = 1KB, RP = 7B, key = 9B & BP = 6B,
Order is?

⇒ 2p(6) + (2p−1)(7+9) ≤ 1024

12p + 32p − 16 ≤ 1024

$p = \left\lfloor \frac{1024+16}{44} \right\rfloor = 23$

PAGE NO. :
DATE : / /

Not a B-Tree } Variation of B Tree

Ques: Since leaf node requires no pointers to children, they could conceivably store a larger no. of keys then internal nodes for the same block size.

Considering this variation of B Tree, assume order of a leaf node is max. no. of keys it can have & order of internal node is max. no.

Block size: 1024B, RP = 7B, key = 9B, BP = 6B., Block header = 10B

a) Order of internal node

$\Rightarrow P(7+9) + (P+1)(6) \leq 1024 - 10$

$\Rightarrow 16P + 6P + 6 \leq 1024 - 10$

$22P \leq 1008$

$P = 45$

b) Order of leaf node

$= P(7+9) \leq 1024 - 10$

$16P \leq 1024 - 10$

$P = 63$

Ques: BSize = 2KB

Pointer = 12B.

Block header = 56B.

key = 8B.

a) Max. no. of records we can index with 3 level B Tree.

» Max. means we have to completely fill the node.

Max. keys in node = n

$n(8+12) + (n+1)(12) \leq 2048 - 56$

$n = 61$ keys

PAGE NO. :
DATE : / /

Max. keys

At level 1 = 61 keys , 62 children

At level 2 = (62*61) keys , (62*62) Children

At level 3 = (62*62*61) keys , $(62)^3$ children

total max. keys = (61) + (62*61) + (62*62*61)

= 238327 keys

b). Minimum keys we need to store to have level 3.

⇒ For min. keys fill the internal node half i.e. Order half. Root with 2 child

i.e. 30 keys (bcz 62/2 ⇒ 31 children)

At level 1 (Root) = 1 key 2 children

At level 2 = (2*30) key (2*31) children

At level 3 = (2*31*30) keys

So min key = (1) + (2*30) + (2*31*30)

= 1921

→ So we can say if order is 62 then if #keys can range from 1921 to 238327 for 3-levels.

c). If we donot want B Tree and allow no DP in leaf node as they are space waste then max & min keys possible for 3 level.

⇒ Here keys in leaf node can be:

$n \in (1278, 2048.56)$

$n = 99$ keys

So max. key in leaf node can be 99.

PAGE NO. :
DATE : / /

At level 1 : 61 keys 62 children
At level 2 : (62*61) keys (61*62) children
At level 3 (leaf) : (62*62*99) keys 0 children

total max keys = (61) + (62*61) + (62*62*99) keys
= 384399 keys ✓

Min keys in leaf node for 50% utilization is 50 keys

At level 1 (Root) : 1 key 2 child
At level 2 : 2*30 keys 2*31 child
At level 3 (leaf) : (2*31*50) keys 0 child

total min. keys : 1 + (2*30) + (2*31*50) keys
= 3161 keys

→ Height of B Tree = #levels - 1.

*) If ques asks for given height, max. no. of keys that can be stored then we fill all levels completely.

Ex: Order = P, height : h

level	#nodes at level	#BP	#keys (max.)
0	1	P	P-1
1	P	(P*P)	P*(P-1)
2	P^2	(P^3)	$P^2*(P-1)$
:	:	:	
h	P^h	P^{h+1}	$P^h*(P-1)$

PAGE NO. :
DATE : / /

total keys = $(p-1) + (p*(p-1)) + (p^2*(p-1)) + \ldots (p^h * (p-1))$

$= (p-1)(1+p+p^2+\ldots p^h)$

$= \frac{(p-1)(p^{h+1}-1)}{(p-1)}$

$= p^{h+1}-1$ ✓

Don't Byheart, just use intution

Max. Nodes = $\frac{p^{h+1}-1}{(p-1)}$ ⇒

* ⇒ If que asks for given height, min. no. of keys that can be stored then have 1 key in root & min. keys in internal & leaf nodes.

Ex: order = p, height = h, let $t = \lceil p/2 \rceil$

Level	Nodes at level	BP	keys
0	1	2	1
1	2	(2*t)	2*(t-1)
2	2t	$2t^2$	$2t^*(t-1)$
⋮			
h	$2t^{h-1}$	$2t^h$	$2t^{h-1}(t-1)$

total key = $1 + 2*(t-1) + 2t(t-1) + \ldots + 2t^{h-1}(t-1)$

$= 1 + (t-1)2(1+t+\ldots+t^{h-1})$

$= 1 + \left[2(t-1) * \frac{(t^h-1)}{(t-1)}\right]$

total min key = $1 + 2(t^h-1)$

total min node = $\frac{1+2(t^h-1)}{(t-1)}$ ✓

where $t = \lceil \frac{p}{2} \rceil$

PAGE NO. :
DATE : / /

* ⇒ If que asks given no. of ~~trees~~ keys, max height possible then do the reverse engg & fill each node minimally.

max height = min. no. of keys per node

$$1+2(t^h-1) = n$$
$$h = \log_t\left(\frac{n+1}{2}\right)$$ ✓

Eg: If h comes to be 48.3 then it will be 48, not 49 as we can't go to $(49+1)^{th}$ level.

⇒ If que. asks give no. of keys, min height possible then fill each node maximally :)

$$n = (p^{h+1}-1)$$
$$\boxed{h = \lceil \log_p(n+1) \rceil - 1}$$ ✓

If ans comes 48.3 then it will be 49 as we can't accomodate in (49+1) levels.

→ Don't use formula, use your own method, derive only if asked ←.

Qu: If p=23 and each node of B tree is 69% full then min no. of keys in nodes are:

⇒ (23*.69) BP = 16 BP

i.e. 15 keys ✓

PAGE NO. :
DATE : / /

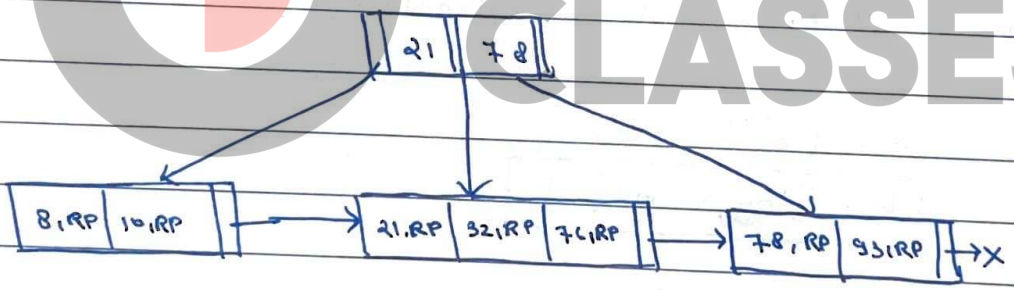
★ B^+ Tree:

★★ Diff b/w B & B^+ Trees:

a) Each data entry (ie. key + record pointer) is present at the leaf node, even if internal node already have them. because in B^+ tree internal nodes are used only for searching, they do not have any Record pointer.

b) Each leaf node has an additional pointer, which is used to move to next leaf in sequence (useful in range query).

Ex: of B^+ Tree



In B Tree some key in leaf and some in non leaf (mutually exc.), but in B^+ tree, some key in nleaf and all key in leaf.

→ In B^+ Trees also:

#leaf nodes = 1 + #key in non-leaf nodes

due to same reason as of B Tree.

PAGE NO. :
DATE : / /

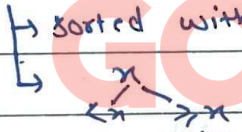
→ Inorder successor of non-leaf key is same key but in an unique leaf node.

* → Order of B^+ tree if not given in que is Maximum no. of total pointers per node.
(generally)

→ Order of leaf node ≠ Order of Non-leaf node.

** Rules of B^+ Tree:

a) Obvious search rules
- sorted within node
- Equal due to Right biasing



b) All leaves are on same level

c) If order is P, then 50% space utilisation is:

Root : 2 BP to P BP

Non-leaf (except root) : $\lceil P/2 \rceil$ BP to P BP

Leaf : $\lceil \frac{P-1}{2} \rceil$ to P-1 ~~keys~~ BP (bcz. 1 BP is fixed)

So $\lceil \frac{P-1}{2} \rceil + 1$ to P total pointers

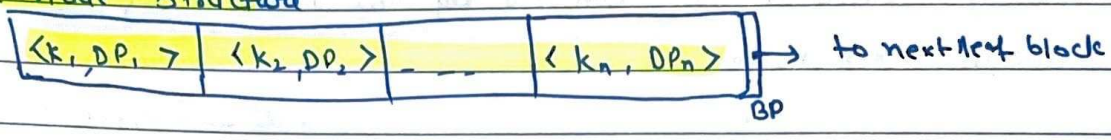
In leaf # R.P = #keys, and 1 BP is fixed

So for 50% utilization $\lceil \frac{P-1}{2} \rceil$ data pointers

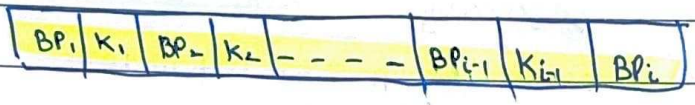
So $\lceil \frac{P-1}{2} \rceil + 1$ total pointer minimum.
($\lceil \frac{P-1}{2} \rceil$ → D.P, $+1$ → B.P)

PAGE NO. :
DATE : / /

d) Leaf node structure



e) Internal Node structure:



→ Root node's structure will be like leaf if it is in one block only else it's structure will be like internal node.

★ Min & Max keys: (let order = p for Internal & $\bar{p}$ for leaf)

	Min key	Max key	Min Ptr	Max Ptr
Root	1	p-1	2	p
Non-Root, Non-leaf	$\lceil \frac{p}{2} \rceil - 1$	p-1	$\lceil \frac{p}{2} \rceil$	p
Leaf	$\lceil \frac{\bar{p}-1}{2} \rceil$	$\bar{p}-1$	$\lceil \frac{\bar{p}-1}{2} \rceil + 1$ (DP, BP)	$\bar{p}$

NOTE ↔ If size of data (or record) pointer = size of block pointer then the order of leaf & internal nodes will be same i.e $p = \bar{p}$

PAGE NO. :
DATE : / /

Qus: If it's given that B+ tree has order p and BP < RP then which blocks are not fully utilized?

⇒ Non-Leaf node blocks must be having some free space.

↳ Ans → Internal Nodes ✓

Qus: Block size = 4096 B, key = 4 B, RP = 9 B, BP = 8 B, then find:

a) Order of internal node, p

⇒ $p(8) + (p-1)(4) \leq 4096$

$p = 341$

b) Order of leaf node, $\bar{p}$

⇒ $(\bar{p}-1)(9) + (\bar{p}-1)(4) + 8 \leq 4096$ (8 → BP to next leaf)

$\bar{p} = 314$

→ In B-tree if height was 10, then we would not go till leaf always, search could be stopped in between i.e. $O(h)$, not $\Theta(h)$

→ In B+ tree, we will always have to go till leaf. So search: $\Theta(h)$ ✓

→ Range search is efficient in B+ trees.

PAGE NO. :
DATE : / /

** B+ Tree searching algorithm:

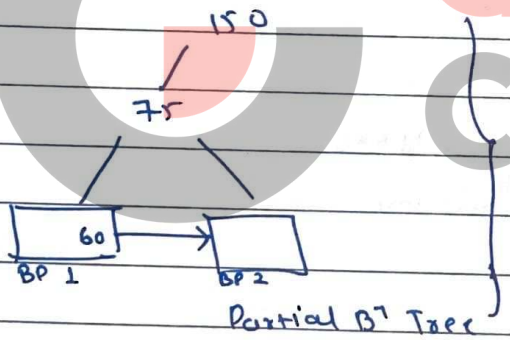
→ For exact key values: (age = 20)

- Start at the root
- Come till leaf and get R.P.

→ For range queries: (20 ≤ age ≥ 30)

- Start at root
- Come till leaf
- Sequential traversal of leaves.

NOTE



Partial B+ Tree

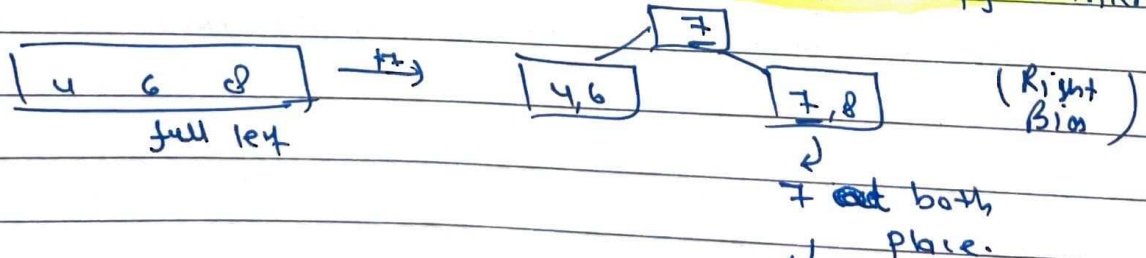
In this B+ tree (partially drawn) if we want to see 30 exists then we don't need BP2 access, as it will not have anything less than 75.

** B+ Tree Insertion:

Similar to B Tree insertion i.e start from leaf and all., except leaf node split. (Internal node split is same)

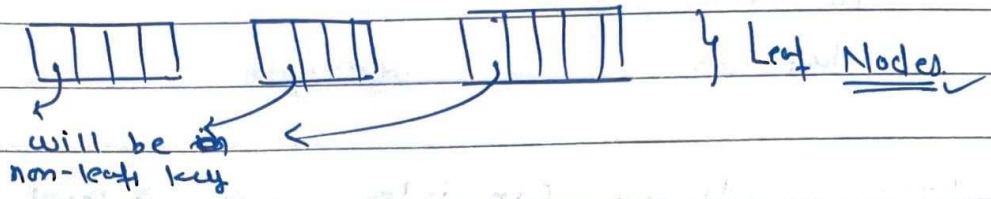
For leaf node we keep it in leaf & copy to internal

Ex:



PAGE NO. :
DATE : / /

Observation: key k will be in non-leaf node if it is splitting point i.e. for Right bias:

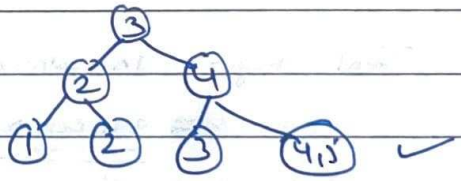


→ Among non-leaf nodes, no key repeats, bcz we on split we are not duplicating the key.

→ Sequence in which keys are inserted matter, same keys in dif seq. can give B trees of dif height also like B Tree.

Ex: Order: 3
Seq: 1, 2, 3, 4, 5

Seq: 2, 4, 3, 1, 5



Qw: Every value of leaf node is also present in non-leaf node in B+ tree?

⇒ False, only some are present.

→ All searches (successful or unsuccessful) in B+ tree take exactly same amount of Disk block accesses. $\Theta(h)$

→ Given order def. and details we can easily find, order of leaf & internal nodes.

PAGE NO. :
DATE : / /

Ques: Block size = 2kB
Pointers = 12B
Block Header = 56B
key = 8B.

Q. Max. records we can index with 3 level B+ tree.

⇒ Only the leaf node records are indexed so we need to get them.

Order of leaf = Order of internal node as BP=RP

$P(12)+(P-1)(8) \leq 2048-56B$

⇒ $P = 100$

So, 99 keys at max in a node

now

keys in level 1 = 99
keys in level 2 = 100 * 99
keys in level 3 = 100 * 100 * 99

Total keys = 100 * 100 * 99
⇒ 9900000 ✓

(Don't add all as only 1st level matter)

Qus: Create B+ tree of order 5 by inserting following data in seq:

92 24 6 7 11 8 22 4 5 16 19 20 78

⇒ DIY ✓

PAGE NO. :
DATE : / /

*⇒ If que asks for given height, max. no. of keys can be indexed then fill each level maximally.

Let ~~order~~ max. keys in leaf node = l & max. keys in internal node = i and height = h.

then.

level	nodes	children
0	1	i+1
1	i+1	$(i+1)^2$
.		
h	$(i+1)^h$	$(i+1)^{h+1}$

↳ with these node we can store $(i+1)^h * l$ keys.

*⇒ And if que asks min no. of keys indexed then fill minimally

If i is min keys in internal node &
l is min keys in leaf node & h is height then

$2 * (i+1)^{h-1} * l$

↳ Bcz. 2 is min. ~~of~~ root.

*⇒ If we are given no. of search keys then, maximum & min. height of B^+ tree can be given by formulae in reverse mech. or by filling leaf nodes and then getting to parent nodes

↳ Bottom up insertion (Bulk loading).

↳ Heap
↳ B^+ Tree

PAGE NO. :
DATE : / /

Que: For same block size & # search keys, which index B or B+ tree will take less no. of levels.

⇒ B+ Tree obviously bcz its order will be more & hence less no. of levels.

Que: If order of B & B+ tree is same then?

⇒ Then B tree will have less levels as it don't have duplicate key in blw.

→ B+ tree is best for Range query.

Last Module, finally :)

M6 - Transaction Management & Recovery.

PAGE NO. :
DATE : / /

→ In DBMS, we do data access, i.e., primitive operations on low level are read from disk & write to disk. We read data from disk into memory, modify it locally in mm & write back to the disk.

→ When we want to update some data entry we read the whole block from Disk & then write back to disk after updating.

→ Transaction in simple terms is a sequence/series of read, write operations on data items.

Ex:
```
Read(A);
A = A+10
Write(A)
```
} A Transaction

In actual we read in temp var and then write that, but for conv. we directly write :

```
Read(A,t)          R(A)
t = t+10     ⇒     A = A+10
Write(A,t)         W(A)
```
(for conv.)

→ Intuitively, our single logical task (like transferring an amount from an acct to other acct., or booking a ticket) on DB correspond to a transaction.

* So, Transaction is a seq. of Read/write actions which makes one logical task.

PAGE NO. :
DATE : / /

We study transactions here because millions of concurrent users may be executing DB trans. concurrently / simultaneously. These transaction require high availability of data and fast response time for millions of users.

We'll see concepts that are needed in transaction processing & concurrency control.

→ Concurrent execution of various transactions which share some data item of DB can lead to inconsistency.

→ DBMS has two components which address respective issues:

a) Recovery Manager: (Atomicity & Durability) Data must be protected in case of a system failure.

b) Concurrency Controller: (Isolation) Data must not be corrupted due to several queries done at once.

★ Some terminology:

→ Database Element: (Data-Item) It is a value which can be accessed or modified by transactions. Element can be Relation, tuple, attribute, disk block or anything else.

From transaction processing POV, database is basically represented as collection of named data-items. Concepts we discuss here are independent of data item granularity (size) & apply to data items in general.

PAGE NO. :
DATE : / /

Each data item has a unique name like if it is disk block, then address, or if it's tuple then record id, etc.

→ **Database State:** Value of each of the data item i.e. Database Instance a snapshot.

State of DB can be:

a) **Consistent State:** DB is consistent if it don't violate:

i) Explicit Constraints | Integrity Constraints like PK, FC, etc.

ii) Implicit Constraints | App. dep. Constraints, that are "in mind" of app. developer.

Many Ex:

↳ Ex: 1) no seat should be given to more than one person
2) Some of two act. must be same before & after the transaction

b) **Inconsistent State:** DB is said to be in inconsistent state if it violates any of the constraint.

→ **Database Operations:** Actual working of opⁿ we'll see in Recovery manager chapter. Here we'll see simplified def. which will only effect efficiency, which don't matter here.

a) **Read (A, t):** 1) Block containing Data item 'A' comes in mm buffer.
2) t = A.
= R(A) when A = A;

b) **Write (A, t):** 1) Block containing A comes in mm buffer
2) A = t
3) Store back the disk block to disk
= W(A) when A = A;

PAGE NO. :
DATE : / /

Assumption:

→ Every transaction has its own mm buffer.

→ Write Op^n updates block into disk immediately.

→ (one after other)

→ If we do serial execution of transactions then if DB was initially in Consistent state then it will surely end in consistent state. But in non-serial exec. we can not always guarantee this.

Ex: Consistency req: A=B,

T_1	T_2
R(A)	
A=A+100	
W(A)	
R(B)	
B=B+100	
W(B)	
	R(A)
	A=A×2
	W(A)
	R(B)
	B=B×2
	W(B)

In this serial execution of transactions, if initially A=B, then after exec. A=B.

Ex: Initially A=B=25

Final: A=B=250

T_1	T_2
R(A)	
A=A+100	
W(A)	
	R(A)
	A=A×2
	W(A)
	R(B)
	B=B×2
	W(B)
R(B)	
B=B+100	
W(B)	

In this non-serial exec. we'll end up in incons state

Ex:- Initially: A=B=50

Final A≠B

2 4

240 ≠ 140 ✓

PAGE NO. :
DATE : / /

→ In simp. def. of R(A) & W(A) we assumed diff buffer for each trans but it might not be the case in Reality and Recovery Mgr decided when to write back the disk block back in disk, instead of immediately writing back. (Don't worry, seen later).

NOTE ↔ write(A), don't read the initial/original value of A it reads the disk block to write the value of A into it w/o seeing original value.

★ Transaction.

→ It refers to the collection of op^n that form a single logical unit of work. It is the unit of execution of DB operations.

→ In addition to R & W, transaction must specify its last action either commit (i.e. successfully complete) or abort (i.e. undo all the actions done so far).

If last action is not written then by default it is commit.

★ Advantages of Concurrent Execution of Transaction

→ Increased processor & disk utilization, so more throughput.

→ Reduced avg. response time.

→ We assume xing CPU system so concurrent exec. is Interleaved execution.

→ If we don't take care concurrent exec. may lead us to inconsistent state.

PAGE NO. :
DATE : / /

⇒ **Concurrency Control:** DB must go from a consistent state to other consistent state after concurrent execution of transactions.

Imp. Q

→ Serial exec. (one trans. after other i.e. no interleaving) always preserve Database Consistency. (Reason - Consistency) ✓ seen below

☆ Desirable Properties for Transaction Processing (i.e. ACID properties)

A. Atomicity- (All or None) It is essential that all the opⁿ of transaction occur or in case of failure, none occur. (You won't want money debit, but not credit)

If a transaction begins to execute but fails for whatever reason (whatever), any changes to the database that the transaction may have made must be undone. This all or none property is Atomicity.

B. Durability (Permanent Changes): After commit, all the changes made by Tran. may not be stored in Disk (bcz. recovery mgr decides to store or not), and any failure occurs (as, disk, any), then durability says that changes must not be forgotten.

The changes applied to the database by a commited transaction must persist in the database, these changes must not be lost because of any, any failure.

PAGE NO. :
DATE : / /

C. Consistency (No coding error) (Correctness Principle)

This states that particular transaction must be correct. i.e. Coding must not be wrong.

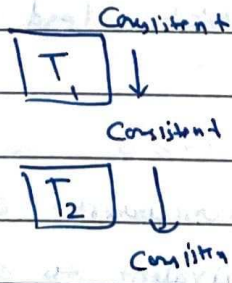
for Ex: A=A-50, B=B+60 } This transaction itself is wrong, so how will DB go to consistent state.

This property is the responsibility of programer / developer.

The responsibility of programmer who codes a transaction to make sure that ALL the DB Consistency req. are satisfied by the transaction.

DBMS can't enforce Consistency of transaction. We will have to trust the developer. So we assume this property. Some authors assume it in Transaction definition.

→ Now coming to our stmt XY2 of prev. page, Serial exec. is always consistent given that Consistent property is there.



D. Isolation (Pseudo feel of being alone): It says that a trans must not see the changes made by other transaction.

There are many ways to ensure Isolation, strongest is Serializability.

PAGE NO. :
DATE : / /

Property	Responsibility of :
Atomicity	→ Recovery Manager bcz. related to failure. (DBMS)
Durability	
Concurrency	→ Programmer
Isolation	→ Concurrency Controller

→ Ex: Violation of Isolation:

#time	T_1	T_2
1	R(A)	
2	A=A-1	
3		R(A).
4	W(A)	
5		R(A)

At #time t_1 T_2 will get A as x and at #time t_2 it will get A as x-1, so it will think that it's not alone & isolation breached.

⇒ ways to ensure isolation:

a) Serial Execution : Extreme strict, lead to poor throughput.

b) Serializability: Allow those concurrent execution whose effect is equivalent to some serial execution
(in detail next)

→ Once every trans. is consistent (C property) then everything is DBMS's responsibility.

PAGE NO. :
DATE : / /

Ques: Is following trans consistent for constraint A=B

A = A*2

B = B*2

→ Yes, Initially A = x , B = x (A = B)

A = 2x , B = 2x

(A = B) ✓

✓ Always check by taking arb. value else we can't prove like $a^2 = a$ will be correct if you take a = 2 ✓

Ques: Suppose that the consistency constraint on the database is $0 \le A \le B$. Tell which of the following transaction preserve it:

a) A : A+B

B : A+B

⇒ A = x , B = y , $0 \le x \le y$

then A = x+y

B = x+2y

now $0 \le x+y \le x+2y$ ↳ True Guaranteed ✓

So Consistency Preserved.

b) B : A+B

A : A+B

⇒ A = x , B = y , $0 \le x \le y$

then B = x+y

A = x+2y

now $x+2y \le x+y$ is not guaranteed

Ex: x = 2 , y = 4 $0 \le x \le y$

& $x+2y \not\le x+y$

So, Unpreserved Consistency.

c) A = B+1 ; B = A+1

DIY. ✓

PAGE NO. :
DATE : / /

Our goal for subsequent lecture is to preserve isolation of transaction. We will see ways to do so.

⇒ Schedule of Transaction: When transactions are executing concurrently in an interleaved fashion, then the order of execution of operations from all the various transactions is known as schedule (or history).

Ex: T_1: a_1; a_2

T_2: b_1; b_2

Then Possible schedule are:

S_1	S_2	S_3	S_4	S_5	S_6
a_1	a_1	b_1	a_1	b_1	b_1
a_2	b_1	b_2	b_1	a_1	a_1
b_1	a_2	a_1	b_2	a_2	b_2
b_2	b_2	a_2	a_2	b_2	a_2

Serial Schedules

Note that opⁿ within a trans. can't be done out of order so a_1 b_2 b_1 a_2 is not a schedule.

→ within a transaction order of opⁿ must not change.

PAGE NO. :
DATE : / /

Que: $T_1 : a_1 a_2 \_ \_ a_n$

$T_2 : b_1 b_2 \_ \_ b_m$

$\vdots$

$T_x : p_1 p_2 \_ \_ p_k$

a) Possible schedules = $\frac{(n+m+\_\_+k)!}{n! * m! * \_\_ k!}$

↳ Use division rule all as b_s p_s same so that we can't reorder them.

Ex: $T_1 : a_1 a_2$ # schedules = MI $\Rightarrow$ 4! ways for all $\Rightarrow \frac{4!}{2!*2!} \Rightarrow 6$

$T_2 : b_1 b_2$

a_1 can't a_2 ; b_1 & b_2 can't reorder

MII $\Rightarrow {}^4C_2 * 1 * {}^2C_2 * 1 \Rightarrow 6$

b) Serial schedules = $x!$ as # Transaction

Ex: for 3 Transaction 6 serial schedules.

c) Concurrent schedules:

$$\frac{(n+m+\_\_k)!}{n! * m! \_\_ * k!}$$

→ Every serial schedule is also a concurrent schedule.

d) Non-Serial schedules:

$$\frac{(n+m+\_\_)! \_\_ (x!)}{n! * m! \_\_ * k!}$$

PAGE NO. :
DATE : / /

→ Serial schedule always preserve consistency. Concurrent schedules may or may not preserve consistency, but we still need concurrent schedules for better throughput. (How? You know). ↙ From OS.

★★ **Serializability:** One of the way to ensure Isolation for concurrent execution.

★ Concurrent schedule S is said to be serializable for Every initial consistent DB state, effect of S is same as some serial schedule.

for Ex:

T_1	T_2	A	B
		x	
R(A)			
A = A+100		x+100	
W(A)			
	R(A)	x+100	
	A = A*2	2(x+100)	
	W(A)		
R(B)			y
B = B+100			y+100
W(B)			
	R(B)		y+100
	B = B*2		2(y+100)
	W(B)		

C. Req: x=y

Initially : x=y
Finally : x=y

} Consistency Preserved by Concurrent Schedule it is eq. to serial schedule $T_1 \rightarrow T_2$

PAGE NO. :
DATE : / /

→ Every serial schedule transforms any consistent database state to another consistent state, but every serial schedule, for same initial DB state will not lead to same final state.

So, serial schedule guarantees the consistency, but not mandatory to have the same state.

Qu: For two transactions T_1 & T_2 assume that we have a schedule S, State T/F?

a) Swapping two consecutive operations of S necessarily makes it invalid schedule on T_1 & T_2.

⇒ false, Not necessarily, may or may not.

Ex: $a_1 b_1 a_2 b_2 \Rightarrow a_1 a_2 b_1 b_2$
valid ✓ valid ✓

b) Swapping two non-consecutive operation of S necessarily makes it invalid schedule on T_1 & T_2.

⇒ True,

Case I: Non-consec. are of diff trans.

$a_1 a_2 b_1 b_2$ ⇒ Invalid.
(due to b_1)

Case II: Non-conse are of same trans

↳ Invalid obviously

c) If Transaction were 3 then?

⇒ Then it can be valid (possible)

Ex: $a_1 a_2 b_1 c_1 b_2 c_2 \Rightarrow a_1 b_1 c_1 a_2 b_2 c_2$ ✓

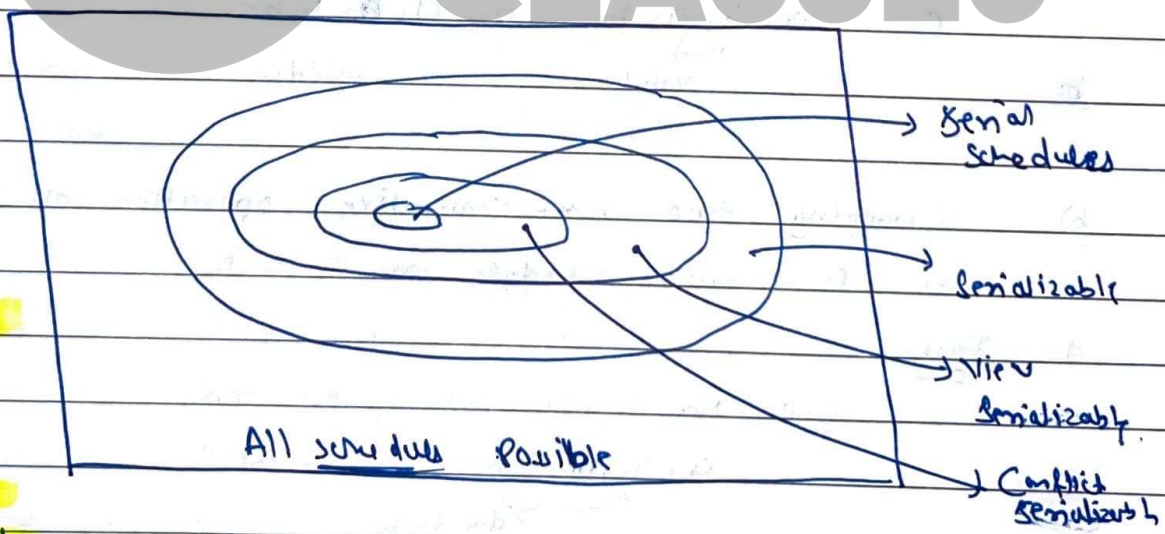
PAGE NO. :
DATE : / /

→ In order to get max. throughput as well as isolation, we execute concurrent schedules which guarantee serializability.
Now, we'll see various Serializable Schedules

a) Serializability (in gen)
↳ Hard or impossible to implement

b) Conflict Serializability
↳ Easy to implement, but less schedules

c) View Serializability
↳ NP Complete i.e. Exponential time Implementation



↑ Area
↑ # of sch
↑ Hard to implement

→ A schedule is serializable if it is equivalent to some serial schedule.

→ Some schedules are serializable but not view serializable because of Computations only.

PAGE NO. :
DATE : / /

→ Serializability in gen. relies on opⁿ & comp. & transaction is a program & what program does is undecidable so serializability (in gen.) can't be implemented.

So, we gen. don't rely on calculation, just see read and write operations, due to which some serializable schedule are lost (but they were hard to know).

Such serializable schedule which only are based on R/W opⁿ & are (& not on computations & actual values)
- Conflict Serializable
- View Serializable

(ignore them) R(A) comp. using A W(A)

★ Conflict Serializability:

⇒ Conflict Pair: We say that two operations I & J in a schedule, conflict if they are operations:

a) By different transactions AND
b) On the same data item AND
c) At least one of them is write opⁿ.

T_1	T_2
R(A)	
	W(A)

T_1	T_2
W(A)	
	R(A)

T_1	T_2
R(A)	
	W(B)

T_1	T_2
R(A)	
	W(A)

T_1	T_2
R(A)	
	R(A)

Conflict Pairs

Not conflict pairs

T_1	T_2
W(A)	
	W(A)

T_1	T_2
W(A)	
	R(A)

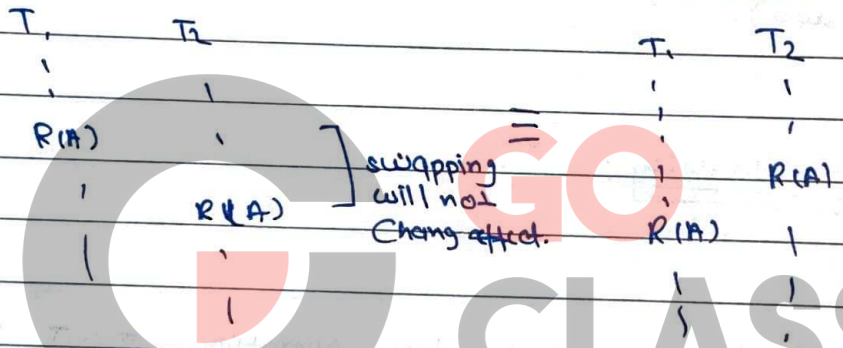
PAGE NO. :
DATE : / /

→ So opⁿ I & J do not conflict, if
a) Both belong to same transaction OR
b) Both are read operation OR
c) They are on d/f data items.

Idea in CS

→ If we swap two consecutive non-conflicting operations, of two d/f transactions in a schedule, it will not change the effect on schedule on the database.

Ex:



→ Conflict Equivalent: If a schedule S can be transformed into a schedule S' by a series of swaps of non-conflicting consecutive instructions of two d/f transactions, we say that S & S' are conflict equivalent.

→ Conflict Serializable: If this above S' is some serial schedule then S is conflict serializable.

A schedule is conflict serializable if it is conflict equivalent to some serial schedule.

PAGE NO. :
DATE : / /

NOTE ↔ We can not swap non-consecutive non-conflictive Op^n of two d/f transactions. Swapping non-consec. Op^n in any schedule gives an invalid schedule. So it is not even schedule if non-consec. Op^n are swapped.

Ex: following schedule are conflict eq. to serial schedule $T_1 \to T_2$, so are conflict serializable

S_1

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

S_2

T_1	T_2
r(A)	
W(A)	
	r(A)
r(B)	
W(B)	
	W(A)
	r(B)
	W(B)

also S_1 & S_2 are conflict equivalent.

Ex: following schedule is NOT conflict serializable:

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

Que: Is the following schedule conflict serializable?

T_1	T_2	T_3
R(A)		
	W(A)	
R(B)		
	R(C)	R(C)
W(B)		
W(A)		
		W(C)

$T_2 \to T_1$ (conflict so, $T_1 \to T_2$, T_1 must be before T_2)

$T_2 \to T_3$

$T_1 \to T_2$ & $T_2 \to T_1$ is impossible

So, NOT conflict serializable.

PAGE NO. :
DATE : / /

NOTE ⟷ Imagine aborted transaction were NEVER there in the beginning, ignore them completely. Only consider commited transactions for serializablity.

Eg:

T_1	T_2
R(x)	
	W(x)
W(x)	
commit	Abort

This schedule is conflict serializable as T_2 is IGNORED

★★

★ **Conflict Serializable test by Precedence Graph:** This is an easy test for conflict serializablity & to get an eq. serial schedule.

Idea is:

Schedule : ____ $R_1(A)$ ____ $W_2(A)$ ____

then T_1 ___ T_2

We draw predecedence graph where each node represent a transaction and edge means it must occur before due to conflicting operation.

$T_1 \longrightarrow T_2$ means conflict op^n b/w T_1 & T_2.

★

→ **Idea of Conflict Serializability:**

When a pair of cnf. action appear in schedule S, the transaction performing those action must appear in the same order in any conflict eq. serial schedules as the action appear in S.

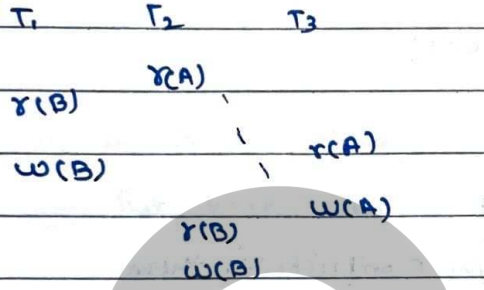
Thus, conflicting pair of action put constraint on the order of transaction in the hypothetical conflic eq. serial schedule. If these constraint are not contradictory, we can find a conflict eq. serial schedule else if they are contradictory, then no such serial schedule exist.

PAGE NO. :
DATE : / /

To tell whether a schedule S is conflict serializable, construct the precedence graph for S & ask if there are any cycles. If so, then S is not conflict serialisable, else it is and order of transaction is topological order of this D.A.G.

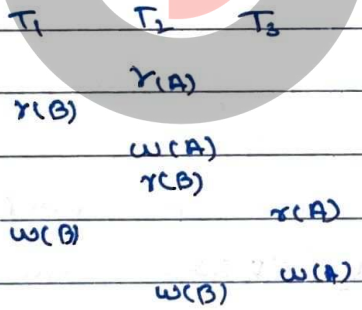
Practice

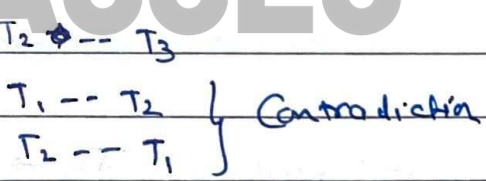
Ex:1



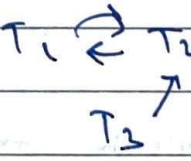
So $T_1 \rightarrow T_2 \rightarrow T_3$

Ex:2





Not Conflict Serializable



Cycle in the graph.

NOTE If there are commits in b/w transaction; you ignore them (commit op^n of aborted transaction) for serializability

misconception: No edge from commited transaction to newly started transaction.

PAGE NO. :
DATE : / /

for Ex:

T_1	T_2	T_3	T_4
	R(x)		
		W(x)	
		Commit;	
W(x)			
Commit;			
	W(y)		
	R(z)		
	Commit;		
			R(x)
			R(y)
			Commit;

$T_3 \rightarrow T_4$ → Yes, conflict is there

$T_2 \rightarrow T_4$

$T_2 \rightarrow T_3$

$T_2 \rightarrow T_1$

$T_3 \rightarrow T_1$

$T_3 \rightarrow T_4$

$T_1 \rightarrow T_4$

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

→ conflict serializable.

→ **Time Complexity** to check conflict serializability:

a) Create precedence graph: $O(n^2)$ time (n → # op^n in schedule)

b) Check cycle : $O(V+E)$ time (V → # Transaction, E → # edges)

$n > V$ in gen.

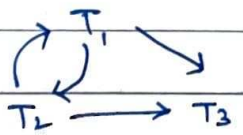
So. $O(n^2)$

PAGE NO. :
DATE : / /

View Serializability: There are some schedules which are serializable (even w/o considering computations), but not conflict serializable

For Ex:

T_1	T_2	T_3
W(Y)		
	W(Y)	
	W(X)	
W(X)		
		W(X)



Precedence Graph

So, it is clearly not a conflict Serializable Schedule But here all writes are Blind writes. So if we do them in order $T_1 T_2 T_3$, then it will have same effic. So it is eq. to serial schedule $T_1 \to T_2 \to T_3$. So, schedule is serializable but not conflict serializable.

→ CS is not necessary but sufficient for Serializability.

→ Idea of view equivalence is, "Always have the same view."

So, every read in the two schedules (view eq.) must have & must give the same view.

Ex:

S_1		S_2	
T_1	T_2	T_1	T_2
R(A)			W(A)
	W(A)	R(A)	
	R(B)		R(B)

View is To

View is T_2

$S_1 \neq_v S_2$

So not view eq.

T_0 → Initial DB Transaction
T_f → Future Transaction.

PAGE NO. :
DATE : / /

Blind write: writing w/o Reading

Ex:2 S_1

T_1	T_2
W(A)	
	R(A) ↓ view(T_1)
W(B)	
	R(B) ↓ view(T_1)

S_2

T_1	T_2
W(A)	R(A) ↓ view(T_0)
W(B)	
	R(B) ↓ view(T_1)

So $S_1 \neq_v S_2$

→ So, schedule S_1 & S_2 are view equivalent iff each read in all transaction have same view in both schedules & final values written by both schedules is same.

Ex:3 S_1

T_1	T_2
W(A)	
	W(A)

final write

S_2

T_1	T_2
	W(A)
W(A)	

$S_1 \neq_v S_2$ ↓ bcz in som T_f if we read B < $S_1 \rightarrow T_2$ / $S_2 \rightarrow T_1$ } somot eq.

Ex:4 S_1

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

S_2

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

$S_1 \equiv_v S_2$ ✓ bcz. both read same A & final write is also same.

PAGE NO. :
DATE : / /

Ex:-

S_1		S_2	
T_1	T_2	T_1	T_2
W(A)		W(A)	
	R(A)		
W(A)		W(A)	R(A)

$S_1 \not\equiv_v S_2$, because, However both are reading A from T_1 but after diff writes, so take care of this also.

* → Schedule S_1 & S_2 are said to be view eq. if ~~two~~ conditions hold :-

~~(1)~~ "for each data item Q, if each transaction in both schedule must read same value of Q. and final write of Q must be from same transaction."

→ A Schedule is said to be View Serializable if it is view equivalent to a serial schedule.

Ex:-

S		
T_1	T_2	T_3
R(A)		
	W(A)	
W(A)		
		W(A)

It is view eq. to $T_1 \to T_2 \to T_3$.
(Initial Read, Final write)

Element.	Reads	w'n	final w'n
A	T_1 from Init	T_1, T_2, T_3	D

So $T_1 - T_2$, $T_1 - T_3$ & $T_2 - T_3$ ✗

$T_1 \to T_2 \to T_3$ ✓

PAGE NO. :
DATE : / /

→ Note that we can directly focus on transaction only if, no transaction writes any element more than once.

Ex:2 T_1 R(A) W(B)

T_2 R(B); W(A); W(B)

T_3 R(A) W(B)

⇒ T_2 -- T_3

T_1 -- T_3

T_2 -- T_1 } Due to W(A); R(A) &
T_2 -- T_3 } also due to R(B); W(B)

So $T_2 \rightarrow T_1 \rightarrow T_3$ is VSR.

NOTE ↔ To check for view serializability:

a) Handle final write of each element.
b) Handle each read operation

} Equation → Contradiction → No VSR
→ No Contradiction → VSR

Ex:3

T_1	T_2	T_3
R(A)		
	R(A)	
		R(A)
W(B)		
	W(B)	
		W(B)

Clearly T_1 --- T_3 } Due to final write
T_2 - - T_3 }

So VSR are $T_1T_2T_3$ & $T_2T_1T_3$.

PAGE NO. :
DATE : / /

Ex: 4

T_1	T_2	T_3	T_4
	$r(A)$		
$r(A)$			
$w(C)$		$r(C)$	
$w(B)$			$r(B)$
		$w(A)$	$r(C)$
	$w(D)$		
	$R(B)$		
			$w(A)$
			$w(B)$

⇒

Element	writes	Read	final write.	Equations
A	T_2 T_4	T_3, T_1 ↳ Initial	T_4	$(T_1 T_2)$-$(T_3 T_4)$ $T_3 - T_4$
B	T_1, T_4	T_4 ; T_2 ↳ T_1 ↳ T_1	T_4	$(T_1 - T_2)$; $(T_1 - T_4)$
C	T_1	T_3 ; T_4 ↳ T_2	T_1	$(T_2 - T_3)$; $(T_2 - T_4)$
D.	T_2	—	T_2	

So, from eq:

$T_1 \rightarrow T_2 \rightarrow T_3 \rightarrow T_4$ is VFSS

NOTE ↔ Take care when any transaction write same element more than once.

(VS) (CS)

★★★ View Serializability v/s Conflict Serializability: VS uses the benefits of Blind writes & CS ignores them.

If no blind writes in S, & S_1

then VFSS = CFS.

So only d/f b/w VS & CS is when Blind writes are present.

PAGE NO. :
DATE : / /

→ An View serializable schedule which is not conflict serializable contains a blind write.

→ VS has exp. T.C, but it covers more serializable schedule.

→ Constrained Write Assumption ≡ No blind write
↓
CS = VS

→ Effects of commit and abort are same in both VS & CS.
i.e. ignore commit opn & aborted transaction.

★★★ Serializability v/s. View Serializability:

If only read & write operation are given then
VS ≡ Serializability

If Comp are involved in schedule then some serializable schedule can be non-view serializable schedule.

Eg:

T_1	T_2
R(A)	
t=t+100	
W(A)	
	R(A)
	A=A+200
	W(A)
	R(A)
	A=A+300
	W(A)
R(A)	
A=A+100	
W(A)	

Serializable as additions can be done in any order, but NOT VS.

$T_1 \leftrightarrow T_2$
↓
Not CS
↓
Not VS
bz no blind write.

{ CS → VS & Not VS → Not CS }
↳ if blind write else eq.

PAGE NO. :
DATE : / /

☆ Recoverability

☆☆ Failure & Recovery :

→ So far, we have studied schedule while assuming implicitely that there are not any transaction failures. We now will see the effect of transaction failures during concurrent execution.

→ Serializability ensured us Isolation & Database consistency given that there is not any failure.

⇒ Commit point of Transaction : Transaction T reaches its commit point when all its operations that access the database have been executed successfully.
Beyond the commit point, the transaction is said to be commited and its effect must be permanently recorded in the database (also called Durability property).

→ If a system failure occurs we can search for all transactions T that have not executed the commit opⁿ yet ; these transactions may have to be rolled back to undo their effect on the database during the recovery process. Transaction that have committed, their effect on the database must persist.

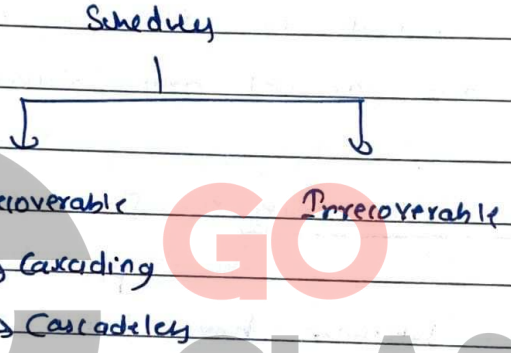
☆→ Our overall goal (hero of movies) is consistency of DB, which may be compromised in case of system failure.
Also atomicity may be compromised. So to ensure atomicity which ensures Consistency we have commit point & we rollback uncommited transactions

PAGE NO. :
DATE : / /

★★ Recoverability of Schedule:

→ For some schedule, it is easy to recover from transaction & system failures, whereas for other schedules, the recovery process can be quite involved. (tough). (Cascading)

In some case, it is not even possible to recover correctly after a failure (Irrecoverable schedules)



NOTE → It is always ensured that commited transaction are not req. to rolled back, so that durability is not violated.

→ Violates Atomicity

★ **Irrecoverable schedule:** Analogy is you are cheating from someones paper & you submit the sheet and he cuts down that answer then there is no way for you to recover. from

Similary if Tx is reading value written by Ty and Tx commit before Ty and Ty aborts, then it is not possible to recover. Such schedule are non-recoverable Schedule.

PAGE NO. :
DATE : / /

For Ex:

T_1	T_2	
W(A)	→ Dependency	Irrecoverable as
	R(A)	↳ T_2 has wrong value
	commit	& T_1 is rolled back so
abort.		Consistency violated.

Here atomicity is affected as if T_1 fails it will be rolled back but still it has affected DB (partially) and atomicity is all or none.

Recoverable Schedule: A recoverable schedule is one where for each pair of transaction T_i & T_j such that T_j reads a data item written by T_i, the commit operation of T_i appears before the commit operation of T_j.

Ex:

T_i	T_{i+1}
R(x)	
W(x)	
	R(x)
commit;	
	commit;

Ques: Is following schedule Recoverable?

Q)

T_1	T_2	
W(A)		Recoverable
	R(B)	as NO dependency.
	commit	
commit;		

PAGE NO. :
DATE : / /

B)

T_1	T_2	
W(A)		
	W(A) → Blind write	Recoverable due to NO dependency. ✓
	R(A)	
	W(A)	
	commit;	
commit;		

C)

Ti	Ti+1	
W(A)		Recoverable. ✓
	W(A)	
	commit;	
commit/abort;		

NOTE ⟷ Serializability and Recoverability are Orthogonal Concepts i.e. Nothing to do with each other. No relation.

→ Atomicity is violated when Ti fails & we are not able to undo ALL effects of Ti. i.e. can't undo all dependent transactions. So to preserve atomicity we can only have recoverable schedules

→ There are two types of Recoverable Schedules.

*** Schedules w/o Cascade Rollback i.e. Cascadeless Schedules (Easily Recoverable)

V/S

Schedules with cascade Rollback i.e. Cascading Schedules (Recoverable but waste of resources)

ACR : Avoid Cascading Rollback Sc

↓

Cascadeless Schedule

PAGE NO. :

DATE : / /

Ex: 1

T_1	T_2	T_3
W(A)		
	R(A)	
		R(A)
abort		
	commit	
		commit

Cascading Schedules

Here with T_1's rollback, cascadeley we'll have to rollback the T_2 and T_3. (Many transactions rollbacking due to one's abort)

Ex: 2

T_1	T_2	T_3
R(A)		
W(A)		
commit/abort		
	R(A)	
		R(A)
	commit	
		commit

Cascadeless Schedules

Even if T_1 fails we'll not Rollback T_2 & T_3., so it's a Cascadeless schedule.

↳ * This type of Schedules just do NOT read from uncommited transactions. (i.e. don't do Dirty Reads)

→ Only principle of Cascadeless Schedule is don't read from uncommited transactions. That's all.

T_1	T_2	T_3	T_4
W(A)			
	R(A)		
commit		W(A)	
	commit	commit	
			R(A)
			commit

Cascading

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

Cascadeless

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

Irrecoverable

PAGE NO. :
DATE : / /

Most restrictive

★ **Strict Schedule:** Don't read/write from uncommited transaction

(Cascadeless said don't read, it even block writing)

Eg:

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

Cascadeless but Not Strict.

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

Strict and Cascadeless.

⇒ Given a schedule, checking if it is conflict serializable or not is EASY, BUT its too LATE.

System must had already done a lot of work, now we are checking its serializablity, if it was serializable or not and undoing if non-serializable. It's a poor & painful way.

Better idea is to ensure serializability, during execution, but how? How will we ensure that system only generates Conflict Serializable schedule during execution only. ?

↳ We'll see now (last thing in DBMS)

PAGE NO. :
DATE : / /

☆ **Concurrency Control Protocols:** Our goal with these protocols is to implement restriction such that ~~protocol~~ schedule generated is:

- Conflict Serializable
- Recoverable and preferably cascadeless.

→ With minimum (overhead) headache we can turn off interleaving, but it will provide a poor degree of concurrency.

→ Concurrency control schemes tradeoff b/w the amount of concurrency and the amount of overhead that they incur.

→ Now, we'll design concurrency control protocols (set of rules) that, each transaction, follows individually, and automatically system only generates Conflict Serializable + Recoverable schedules.

→ There are two categories of Concurrency Control protocols:

a) Lock Based Protocol
   - General
   - 2PL

b) Timestamp based protocol.

☆☆ **Lock Based Protocols:**

A: **General Lock-Based Protocols:** To ensure isolation, these rules ensure that data items can be accessed in a mutually exclusive manner, i.e. while one transaction is accessing a data item, no other can modify that data item.

PAGE NO. :
DATE : / /

Locking Protocol Rules:

→ There are two types of locks, i.e. Data items can be locked in two modes:

a) **Exclusive (X) mode (X-Lock):** If transaction Ti has obtained an X-Lock on data item Q, then Ti can both read & write Q

b) **Shared (S) mode (S-Lock / Readlock):** If a transaction Ti has obtained an S-Lock on item Q, then Ti can read but cannot write Q.

→ Unlock(Q) is used to unlock any type of lock on Q.

→ Every transaction before accessing any item Requests X-lock if it wants to write item or S-lock otherwise It requests lock from Concurrency control Manager which is a part of DBMS S/w.

→

	Already has	Now Asking	Result
1.	Shared	Shared	Granted
2.	Shared	Exclusive	Denied
3.	Exclusive	Shared	Denied
4.	Exclusive	Exclusive	Denied

Lock Mode Compatibility

→ By default we don't allow lock conversion i.e. first acting Shared and then converting it to exclusive lock.
If we want to do so, it must be mentioned.

SL(A) → Shared lock on A
XL(A) → Exclusive Lock on A
U(A) → Unlock A.

PAGE NO.:
DATE: / /

→ These rules will be followed by individual transactions and not schedule.

→ If we allow early unlocking, then it is possible to gen. a non conflict Serializable schedule.

For Ex: T_1: R(A) W(B)
T_2: W(A) R(B)

and schedule generated is:

T_1	T_2
SL(A)	
R(A)	
U(A)	
	XL(A)
	W(A)
	U(A)
XL(B)	
W(B)	
U(B)	
	SL(B)
	W(B)
	U(B)

This is a Non CS schedule.

⇒ So, using general general locking protocol, "Non-Serializable" schedules are possible, so we will now have to put some more Restrictions.

→ Even deadlock is possible, which will lead to failure.

∴ Ex:

T_1	T_2
XL(A)	
W(A)	
⊗	XL(B)
	W(B)
XL(B)	
	XL(A)

T_1 ⇄ T_2 waiting both.

PAGE NO. :
DATE : / /

→ So using general locking protocol, Deadlock possible, starvation possible, inconsistency possible, Non-Serialisable schedule possible.

⚠ Now if we do delayed unlocking i.e. unlock at very last of transaction, then we will surely preserve consistency i.e. get Conflict serialisable schedule only BUT deadlock still possible.

→ We prefer "Deadlock with consistency" instead of "Inconsistency with No deadlock", we can live with deadlock but can't with inconsistency.

So deadlocks are definitely preferable to inconsistent states., since they can be handled by roll back of transactions., but inconsistency can lead to real world problems.

→ Most locking protocol suffer from Deadlock & starvation, but it's okay, we will adjust.

B. Two-Phase Locking Protocol:

B1. Basic 2PL (bydefault 2PL): This protocol requires that each transaction issue lock & unlock request in two phases:

- **Growing Phase:** Transaction may obtain locks (& do other works as well), but may not release any lock.
- **Shrinking Phase:** A transaction may release locks (& do other work) but may not obtain new locks.

PAGE NO. :
DATE : / /

→ Initially the transaction is in the growing phase, it acquires locks as needed, once tranaction releases a lock, it enters the shrinking phase and now it can't issue any lock request.

→ 2PL rules are General locking rules + Two phase rule.

*→ 2PL protocol says that if every transaction individually follows 2PL rules, then every schedule that will be generated will be conflict serializable. Le CESS will be seq. of last lock acq.

~~S~~S(x)	S(x)
U(x)	X(y)
XL(Y)	U(Y)
U(Y)	U(X)
Not in 2PL	In 2PL ✓

→ Note that the unlock instruction need not to appear at the end., it can be in blw, bw can't lock anything once unlocked something.

→ The point at which the transaction in schedule has obtained its final lock (end of growing phase) is called the lock point. And order of lock points of transactions is eq. serial schedule.

→ To check if s can or can not be generated by 2PL, try to generate ~~2PL~~ s by following 2PL rules

PAGE NO. :
DATE : / /

Qu: Can it be produce by 2PL schedule;

A)

T_1	T_2	T_3
		R(D)
W(A)		
	R(B)	
		W(B)
W(D)		
	R(D)	

⇒

T_1	T_2	T_3
XL(A)		
W(A)		
	SL(B)	
	R(B)	
		XL(D)
		W(D)
XL(D)		
W(D)		
U(A)		
U(D)		
	SL(D)	
	R(D)	
	UL(B)	
	UL(D)	

⇒ So, yes and seq will be $T_3 \to T_1 \to T_2$.

B)

T_1	T_2	T_3
XL(A)		
W(A)		
XL(B)		
U(A)		
	XL(A)	
	R(A)	
		W(B)
W(B)		
		W(B)
	W(A)	
		R(B)
	R(B)	

Now what to do?
~~Can't~~
We can't move fwd.

⇒ Can't be produced by a 2PL scheduler.

PAGE NO. :
DATE : / /

NOTE ⟷ 2PL gen. schedule are subset of CS schedules.

i.e. 2PL gen. → CS

CS ↛ 2PL

Not CS → Not 2PL gen.

i.e. there are some schedules that are conflict serializable but cannot be generated by 2PL scheduler.

→ Transaction follows 2PL rules, NOT a schedule. Schedules is 2PL means S can be produced by a 2PL scheduler.

Ex:

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

It is CS but cannot be generated by 2PL.

→ If schedule is not even CS, then no sense in checking 2PL. If it is CS, then check 2PL.

★★ Deadlock x Basic 2PL: Every locking protocol except those using deadlock prevention suffer from deadlock.

So, our basic 2PL also suffers from deadlock.

Ex:

T_1	T_2
XL(A)	
W(A)	
	XL(B)
	W(B)
W(B)	
	W(A)

Deadlock

PAGE NO. :
DATE : / /

*→ Imp. upgrade from Basic locking to 2PL provides us that 2PL guarantees CS of schedules.

*→ Basic 2PL do not guarantee Recoverability. (Can gen. irrecoverable schedule)

Ex:

T_1	T_2
XL(A)	
W(A)	
U(A)	SL(A)
	R(A)
W(B)	
U(B)	SL(B)
	R(B)
	U(A); U(B);
	Commit;
Commit;	

Non-Recoverable Schedule. & it can be generated by 2PL schedules ✓

→ To guarantee recoverability, transactions must not release locks too early.

B2 Strict 2PL: This is variation of 2PL which do not only provide us recoverability, but Avoiding Cascade Recoverability i.e. Cascadeless Recoverability.

It has a simple addition Rule is, "Release X-Lock only after Commit/abort".

↳ So there will be no dirty read/write.

However we only want to avoid dirty read, but it is also avoiding dirty write.

Strict 2PL provides strict schedules by avoiding dirty read and dirty write.

Strict 2PL still suffers from Deadlock.

PAGE NO. :
DATE : / /

B3. **Rigourous 2PL:** It is much more stricter than Strict 2PL. It says release all locks only after commit/abort. Not any specific benefit of this, as it also provides ACR & suffers from Deadlock.

→ In this shrinking phase starts from commit point.

B4. **Conservative 2PL:** It is used to prevent deadlock, it just says, take all locks you want in start of the transaction. You can't request lock in between:

Here Deadlock is not possible.

Now if we release excl. lock after commit point then it is ACR schedule also.

☆ Summary:

	Conflict Ser.	ACR.	Deadlock
Basic locking	X	X	X
2PL (basic)	✓	X	X
Strict 2PL	✓	✓	X
Rigourous 2PL	✓	✓	X
Cons. 2PL	✓	X	✓
Con 2PL + Strict 2PL	✓	✓	✓

→ By default, irrecoverable schedules are allowed in conservative 2PL, but deadlock is not possible.

PAGE NO. :
DATE : / /

→ The size of the data item is called granularity. Granularity of the data items is the portion of the database, a data item represents.

More the granularity, lesser is the concurrency.

Ex: If data item is entire DB, then only serial execution possible.

So, for better concurrency, data items are taken as small item like a Record or an attribute value in Record.

BS: 2PL with lock Conversion: In this we are allowed to upgrade (SL → XL) or downgrade (XL → SL) the lock.

In growing phase only lock upgrade is possible & in shrinking phase only lock downgrade is possible.

Note that for upgrading, no other transaction should have S on that data item.

Ex

T_1	T_2
S(A)	
	S(A)

Upgrade → Not possible

Some schedule not allowed by 2PL(basic) are allowed with lock Conversion.

Ex:

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

} Not allowed by basic 2PL but allowed if you allow lock conversion.

→ It will surely provide CS only.

PAGE NO. :
DATE : / /

→ Lock conversion seems to be a better option (in basic 2PL), but we do not allow lock conversion bcz it increases the chances of Deadlock.

Ex:

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

} Deadlock not possible with 2PL (basic) but possible by 2PL with lock conversion.

∴ lock conversion ↑ CFS but also ↑ chance of deadlock.

⁂ Deadlock Handling:

Deadlock happens when ∃ a set of transactions such that every transaction in this set is waiting for other transaction to release locks.

for Ex:

T_1	T_2
XL(A)	
W(A)	
	XL(B)
	W(B)
wait for B	
	wait for A

} Deadlock.

* **Deadlock prevention:** If we implement this, then deadlock can never occur and we don't have to periodically check for deadlock.

PAGE NO. :
DATE : / /

⇒ Deadlock prevention schemes:

A) Conservative Approach: Take all locks before starting transaction execution. i.e. Conservative 2PL
→ Not Practical } As we don't know in advance.

B) Putting partial ordering on the data item & accessing item according to this order only.

Eg: A 1
B 2
C 3
} Now after accessing C you can't access A & B.

C) Prevention using Transaction Timestamp:

Let Timestamp at which T_i is gen is $TS(T_i)$
then if T_i gen. before T_2
then $TS(T_1) < TS(T_2)$
↓ Older → Younger

Now we should be giving more priority to older transaction, as it must have done much more work.

→ Non-preemptive

C1) Wait - Die Scheme: Old waits & young dies

Say T_i want lock on A but T_j holds lock on A

Case 1: $TS(T_i) < TS(T_j)$
T_i waits for T_j to release locks

Case 2: $TS(T_j) < TS(T_i)$
T_j — T_i dies ✓

Wound → To abort someone else

PAGE NO. :
DATE : / /

do if younger wants lock held by older then younger dies, and if older wants lock held by younger, then older waits.

→ Preemptive

2) **Wound-wait**:

If Ti wants lock on item which Tj already holds lock on.

Case 1: $T_s(T_i) < T_s(T_j)$

Ti will abort Tj (wound) to snatch data item from Tj

Case 2: $T_s(T_j) < T_s(T_i)$

Ti will wait for Tj to release.

So, old in need abort young & young in need wait for old.

→ So in both scheme young dies either suicide or wound by older.

NOTE ↔ If a transaction aborts, it will restart again after starting with OLD TIMESTAMP only. (to avoid starvation)

↳ If keep coming with high TS, again & again killed by older transactions. So make sense to start with old TS.

* **Deadlock Detection**: If we don't implement deadlock prevention (as costly to do so) we can periodically detect the deadlock.

→ Deadlock Detection is done using "wait for" graph.

↳ Deadlock iff cycle. ⇒

(In OS)

PAGE NO. :
DATE : / /

Timestamp based Protocol: (TSBP)

It says, "your fate is already decided, you just act according to it." While the lock based protocol (LBP) said, "your actions decide your fate".

→ In TSBP, serializability order of transactions are determined before they execute on the basis of time they were generated, while in LBP, determination of serial order is based on last lock executed during execution.

→ When a Txn T is created, it's assigned a Timestamp

If $T_1 \neq T_2 \rightarrow TS(T_1) \neq TS(T_2)$

and $TS(T_1) > TS(T_2)$

(T_1: younger, T_2: older)

→ Serializability order is fixed "before execution" in order of Timestamps.

→ Basic rule of TSP is:

$TS(T) < TS(T')$, then order of conflict operation in any schedule b/w T & T' should be in order $T \rightarrow T'$.

→ (TBSP) It is based on simple principle, "If you are late, it is your fault, you will abort".

→ Let $TS(A) < TS(B) < TS(C)$

then A	B	C
R(A)		
	W(A) ✓	
		R(A) ✓

A	B	C
	R(B)	
		W(A)
		R(A)

It is Wrong
A will abort as it's late.

A	B	C
		W(A)
	R(A)	

PAGE NO. :
DATE : / /

A	B	C
	R(x)	
R(x)		
	W(x)	
		r(x)
		W(x)

B is late so B will abort.

→ for every data element we have Read time stamp (RTS) & write time stamp (WTS) which holds the youngest Transaction to perform these opn on these data items. (Value is not updated with older once younger has read)

for Ex: $T_1 < T_2 < T_3$

		RTS	W.D
		X	1
R(x)		2	
		Y	3
W(x)			3
	W(Y)		
W(Y)			
	R(Y)		
R(x)			

→ Now Read rules are if $T_0 \rightarrow T_j$

T_0 wants to read(x) &
T_j has written(x) then abort T_0, bcz T is late.

T_j wants to read(x) &
T_0 has written(x) then no problem

PAGE NO. :
DATE : / /

→ Now write rules is $T_0 \to T_y$ then

if T_0 want to write (x)
& T_y has already read then abort T_0.

if T_0 wants to write (x)
& T_y has already written(x) then abort T_0

if T_y want to write (x) &
T_0 has read or write (x), then no problem.

→ If a transaction aborts in TSBP, then it restarts with a fresh new timystamp, else it will abort again and again. (as it will be always late)

* / Thomas / Write Rule:

→ Rules summarised,

if T want to read x &
TS(T) < WTS(x), → abort T
TS(T) ≥ WTS(x) → fine.
TS(T) ≥ RTS(x) → Always fine

if T wants to write x &
TS(T) < WTS(x) → abort T
TS(T) < RTS(x) → abort T
TS(T) ≥ RTS(x) ≥ WTS(x) → fine.

PAGE NO. :
DATE : / /

→ **Thomas Write Rule:** It is an optimisation of TSBT, where if T_o wants to write & T_y has already written, then ignore write op^n of T_o because final write was going to be of T_y only. [$TS(T_o) < TS(T_y)$]

b, it says $TS(T) < WTS(x)$, then ignore write(x) of older transaction, instead of aborting it.

However if older wants to write and younger has read value, then we have no option other than rolling back.

Ex:1 $T_1 < T_2$

$R_1(B)$
$R_2(B)$
$W_2(B)$
$R_1(A)$
$R_2(A)$
$W_1(A)$

	RTS	WTS
A	✗2	2
B	✗2	2

No problem

Ex:2 $T_1 < T_2$

$R_1(A)$
$W_2(A)$
$W_1(A)$ ✗
$R_1(A)$

→ ignored if Thomas write rule
→ This will always abort T_1

→ By default do not use thomas write rule.

PAGE NO. :
DATE : / /

⇒ In exam, to solve question do not check by RCS & all, they are rules to understand only. To solve the question write transaction from L to R in increasing order of timestamp

And now, check the conflicts, if there are conflicts then abort older transaction. Conflicts backward are not allowed.

Ex $T_1 < T_2$

W(x) → Allowed

R(x) ← Not allowed

W(x)

Only W(x) W(x) backward conflict is allowed by Thomas write rule.

NOTE ↔ TSP ensures conflict serializability, but TSBP with Thomas write rule allows non-conflict serializable (but correct) schedules.

So TBSP with thomas write rule is not subset of C.S.

Ex: $T_1 < T_2$

R(A)

W(A)

W(A)

} Not CS, but allowed by TBSP with thomas write rule.

PAGE NO. :
DATE : / /

Que: Consider the below schedule.

T_1	T_2
R(A)	
R(B)	
R(C)	
	R(A)
	R(B)
	P: ___
Q: ___	
	R(C)

If $TS(T_1) < TS(T_2)$, then what should be P, Q. for which above schedule is allowed under Thomas write rule but not under basic TSBP.

a) W(B), W(C)
b) W(A), W(A)
c) both
d) None

→ a is allowed anyways and b is allowed in none (seems allowed by thomas write rule, but it's not, due to R(A) in T_2).

Que: Is this schedule allowed by 2PL & TSBP:

T_1 < T_2

W(x)

R(x)

=) Allowed by 2PL & serializability order is $T_2 \rightarrow T_1$ but not allowed by TSBP as its fixed ser. order is $T_1 \rightarrow T_2$.

Note

Que:

PAGE NO. :
DATE : / /

Que: In any schedule, the transaction that executes the first data access, has the least timestamp?

⇒ **No.** TS is given to transaction when they are created, not when they do data item access.

Que: How many transactions are aborted in below schedule

$T_1 < T_2 < T_3 < T_4$

r(y) r(x) w(y) w(x) r(z) r(x) w(z). w(y) w(z)

w(x) → due to this w(x), T_2 aborts

w(z). → due to this w(z) T_3 aborts.

No it is not due to this as T_2 already aborted

Ans: ~~3~~ 2 (Not 3)

→ TSBP do not suffer from deadlock, because in TBSP, transactions never wait. But it may suffer from starvation if same transaction is aborted frequently.

→ TBSP do not ensure recoverablity

Ex: $T_1 < T_2$

W(A)

r(A)

commit

abort;

} Allowed by TBSP, but not recoverable

PAGE NO. :
DATE : / /

→ TSP do not ensure CS.

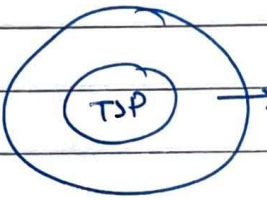
For Ex: $T_1 < T_2$

R(x)

. W(x)

W(x)

} Not CS but allowed in TSP with "Thomas write rule".



← DBMS Done →

These notes are the property of GOClasses. Please do not share them without their consent.

For any questions or issues regarding my handwritten notes, feel free to reach out via email at karan757527@gmail.com or DM me on telegram at @karan757527 (or by clicking here).

I would love to know if you liked my notes. Your feedback and appreciation are invaluable, so don't hesitate to share your thoughts. Click here to connect with me on LinkedIn or you can find me as @agrawalkaran.

Wishing you success on your journey ahead!

- Karan Agrawal