Date

25.08.12

DBMS

(10 marks)

Textbooks :-

Theory ← { • KORTH • NAVATHE

Exercise practise (Solutions) ← [• RAMAKRISHNAN → Queries, Normalisatio

Contents:-

1. ER Model & Integrity Constraints]→ Theory Questions (1 mark questions)
2. Schema Refinement (Normalization) (4)
3. Query Languoge → Relational Algebra (4), → SQL, → Tuple Relational Calculus
4. Indexing & Physical DB Design (2)
5. Transaction & Concurrency Control (2 to 4)

Introduction:-

• Database :- Collection of related data.

• DBMS :- S/w used to manage & access DB in efficient way.

[User] ⟷ [DBMS] ⟷ [DB]   DBMS Interface b/w user & h/w.

★ ~~If the vo~~

★ File System or OS files fail to manage DB if DB is too huge

Limitations of File System :-

(i) Too Complex (too difficult) to access data from DB files.

e.g. University
- Student
- faculty
- courses

[Student1.txt ⋮ StudentN.txt] 500GB

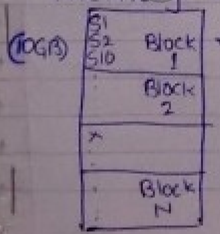
Students who scored >80% → Manually, → Program

→ Location of the file, → Type of the file, → data format } → Physical Details (Storage Details)

(ii) Accessing data using physical details is too difficult.

↳ The DBMS soln. to this is :-
- Hiding physical details to the external user.
- User can access the data without physical details] → Data Independency

(iii) I/O cost :- No. of secondary memory blocks (pages) transferred from secondary memory to main memory in order to access some DB. (The no. of blocks transferred from secondary memory to main memory is the I/O cost.)



← resides in secondary memory.

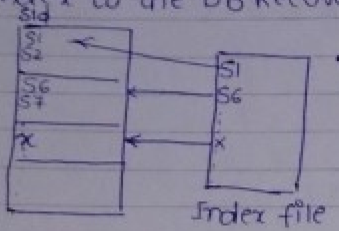
select * from stu where sid = x;

↓

this query is executed when secondary file is first transferred from secondary mem. to main mem. (worst case all N blocks transferred) & then the above query is executed with data of file in main memory.

DBMS soln. to this problem:-

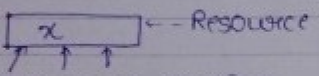
- Index to the DB Records/blocks.



- Now, all the sid's along with the pointers to the block in which corresponding sid resides are stored in index file, so we only need to transfer the index file to the main memory & the query is executed, then the block ~~with~~ which contains 'x' is transferred to main memory (& not all blocks) hence less I/O cost through DBMS.

File System takes more I/O cost, DBMS access the data using ind so that I/O cost is too less.

(iv) Concurrency Control:-

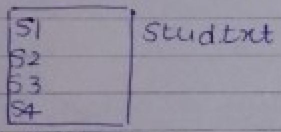


Diff. users accessing same resource simultaneously.

Control Concurrent Accesses:-

- $R_1(x) - R_2(x)$ ✓
- ~~R~~ $R_1(x) - W_2(x)$ X
- $W_1(x) - W_2(x)/R_2(x)$ X

- R_1, R_2 → read by user 1 & respec
- W_1, W_2 → writ user 2 res



U1: Update S1
U2: Update S2

U1	U2
lock (stud txt)	
	lock (Stud txt):- denied because already locked by U1.

(these updates are not affecting each other).

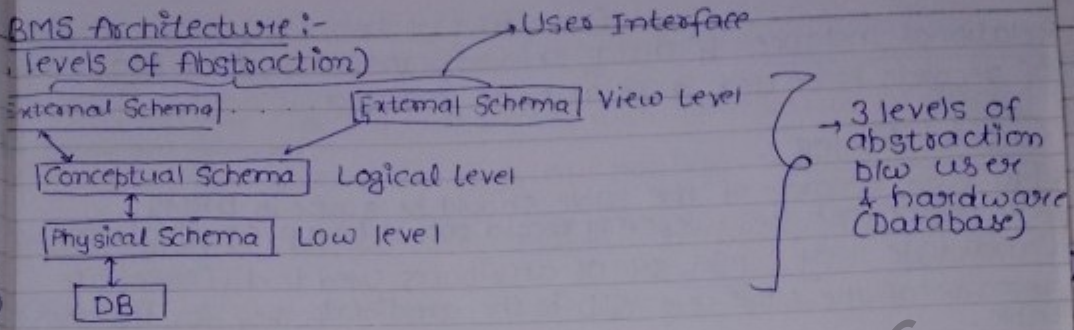
→ but U1 & U2 dont interfere with each other, but even then U1 is locking the complete stud.txt file & hence less concurrency level by OS (less no. of simultaneous access.)

DBMS Soln.:-

- It locks record wise, so U1 will lock only S1 record & not comple stud.txt file, ∴ U2 will be able to access S2 because it is not locked by U1 & hence simultaneous access. concurrency level increa

Concurrency level of record locking is more than concurrency level of file locking.

BMS Architecture :-
levels of Abstraction)



ysical Schema :- Storage details of Database.
t knows how data is physically stored in the database i.e.
→ File Structure
→ Record Structure
→ Field Structure
→ Location/Name/Type of file } → Storage Meta Details (Physical Metadata)

Create Table Student (Sid, .., Sname, ...);
in this case student table is stored in DB & its details like location, size, etc. are stored in physical schema.

Conceptual Schema :- It hides physical details.
t knows what data exists in DB
VIEW (Virtual table) :- data is not physically stored in view.
Every view refers one or more base tables {subset of conceptual schema}.

RDBMS (Relational DBMS) :-
able :- collection of rows & columns.

Sid	Sname	Branch
S1	A	CS
S2	A	CS
S3	B	IT

→ Attributes (or) Field.

rity :- No. of fields of the table. (e.g. 3 in above table), no. of columns.
uple (or record) :- A row in table is called a tuple (or record).
ardinality :- No. of records in the table.
Relational Schema :- Abstract details of table.
Student (Sid, Sname, branch)
Relational Schema.

(records)

Relational Instance:- If data exists in the table, then that se of records is called a relational instance.

Codd Rules:-

- No two records of the table should be same in DBMS. (to imblement this rule, every record should have a candidate key.)

Candidate Key:- Min. set of attributes used to differentiate records of the table. e.g. SID is the candidate key for the abov table.

(Sid, Sname) is not candidate key, because it is not min. set that differentiates two records. (SID can alone do this).

(e.g. if a student can enroll in many courses.

Sid	cid	fee
S1	C1	-
S1	C2	-
S2	C2	-
S2	C3	

in this case (Sid, cid) together forms the candidate key, because sid or cid alone can't differentiate b/w diff. records.

- sid, cid → prime attributes
- fee → non-prime attribute.

★If candidate key forms a single attribute, then candidate key is called simple candidate key, otherwise compound candidate key.

★Attributes belonging to any any candidate key are prime attribu of the relation.

Sid	Sname	PPno	Lno	DOB	fname

Assumes:- No two students with same DOB & fname.

Candidate keys:- {SSno, ppno, lno, (DOB, f

- (DOB, fname) is also cand te key even though SSno is candidate key (which of 1 element) & (DOB, fnam has 2 elements. Min. set of attributes me (DOB, fname) is min. set th can differentiate two reco because DOB or fname al can't do this differentiation

Primary Key:-

One of the candidate key.

(lets take Sid to be the primary key.

- primary key attributes set are not allowed to have NULL values.
- Atmost one primary key is allowed.

ternative Keys:- (Secondary keys)

candidate keys except primary key.

ppno, lno, (DOB, fname)}

NULL values are expected, two records with same value of alternativ keys are not allowed.

more than one alternative keys are possible.

There should be atleast one candidate key with NOT NULL

SUPER KEY:- set of attributes used to differentiate records (min. set not a constraint)

g. if sid can differentiate records, then (sid, sname) is a super key. (sid, ppno) is also a super key, but (sname, fname) is not a super key. me of the subset of superkey must be a candidate key.]

super key attribute Set = Candidate Key Attribute Set + 0 or more other attributes.

Every candidate key is super key, but every super key may not be the candidate key.

g. Student(sid, sname, branch)
{sid}: candidate keys
{sid, (sid, sname), (sid, branch), (sid, sname, branch)}: Super keys

R(A1, A2, ..., AN), How many super keys are possible with

i) only candidate key {A_1} → A_1

no. of subsets possible from A2 to AN → 2^{N-1}

A1 can combine with all these subsets, ∴ total superkeys

2^{N-1} (not +1 because one of the subset is empty, ∴ {A_1} is also included in 2^{N-1}).

ii) candidate keys :- {A_1, A_2}

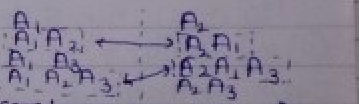
~~A_1~~ A_1 combined with rest others → 2^{N-1}

A_2 " " " " → 2^{N-1}

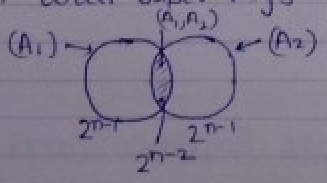
A_1 & A_2 together combined with rest others → 2^{N-2}

∴ total Super keys = $\boxed{2^{N-1} + 2^{N-1} - 2^{N-2}}$

∴-



Some keys are same in both the sets, so we have to delete them once, because these super keys are counted twice.

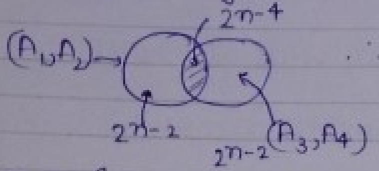


(iii) $\{(A_1, A_2), (A_3, A_4)\} \rightarrow$ candidate keys:-

A_1 combined with rest others $\rightarrow 2^{n-1}$

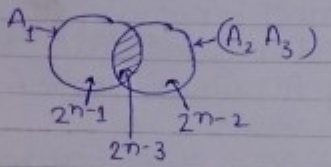
A_2 " " " " $\rightarrow 2^{n-1}$

(A_1, A_2) together with rest others $\rightarrow 2^{n-2}$



$\therefore$ total super keys:- $2^{n-2} + 2^{n-2} - 2^{n-4}$

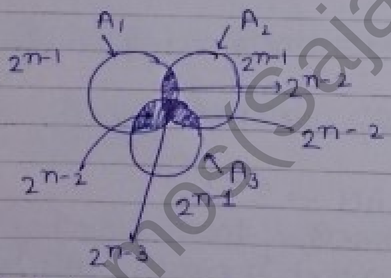
(iv) $\{A_1, (A_2 A_3)\} \rightarrow$ candidate keys



$\therefore$ total super keys:- $2^{n-1} + 2^{n-2} - 2^{n-3}$

(v) $\{A_1, A_2, A_3\} \rightarrow$ candidate keys

due to $A_1 \rightarrow$



☆ $2^{n-1} + 2^{n-1} + 2^{n-1} - 2^{n-2} - 2^{n-2} - 2^{n-2}$

(vi) R(A1, A2, . . ., AN) [candidate key is not given]

How many super keys are possible?

(a) $n!$ (b) 2^n [✓ (c) $2^n - 1$] (d) 2^{2^n}

take example, if we have R(A, B, C)

all sets :- A

B

C

AB

BC

AC

ABC

$\} \rightarrow 2^3 - 1$.

☆ if every attribute of reln. is candidate key, then max $\boxed{2^n - 1}$ superkeys are possible.

chema Refinement (Normalization) :-

Eliminate/reduce redundancy in relations.

edundancy :- Duplite copies of same data.

Redundancy results in wastage of storage space.

If two or more independent reln. ~~ke~~ are kept in same table, then redundancy is always possible.

Sid	Sname	Cid	Cname	fee
S1	A	C1	DS	2K
S1	A	C2	DB	5K
S2	A	C_2	DB	5K
S3	B	C2	DB	5K
S3	B	C3	DS	10K

edundancy

same course id ~~name~~ must have same course name, but diff. cid can have same cname.

(sid, cid)
↓
primary key

Problems because of redundancy

1. Updation Anamoly :- Updation req. in all duplicates copies which is too costly.
2. Insertion Anamoly :- Because of independent details, it is not possible to enter some details without other details, e.g. we can't enter course details for a new course without addition of a student, because we can't put Sid to be NULL as it is a part of primary key.
3. Deletion Anamoly :- Because of deletion of some data, it is possible to loose some other independent data, e.g. deletion of Student S1 course's C1 details, so we delete 1st row, this causes course C1 info to be deleted.

²4:

ecomposition of the relation :-

Splitting relation into two or more relation.

Sid	Sname
S1	A
S2	A
S3	B

Sid	Cid
S1	C1
S1	C2
S2	C2
S3	C2
S3	C3

Cid	Cname	fee
C1	DS	2k
C2	DB	5k
C3	DS	10k

all the anamolies are overcome.

unctional Dependency :-

sid → sname, this means wherever the sid is same, then sname should be same, but not vice-versa.

Let R be the relational schema with X, Y as attribute sets.
X → Y exists in R only ~~if~~ if T_1, T_2 tuples ∈ R such that if $T_1.X = T_2.X$, then $T_1.Y = T_2.Y$.

1. $X \rightarrow Y$ is always true, when X is super key, i.e. if two superk are same, then Y must be same.

X	Y
x_1	y_1
x_1	y_2
x_1	y_1

$X \not\rightarrow Y$

Functional Dependency → Trivial FD, → Non-Trivial FD

Trivial FD:-

X, Y are attributes sets over R
if $X \supseteq Y$ then $X \rightarrow Y$
(Y is a subset of X) or (Y is super set of X)

e.g. sid → sid
sid, sname → sid
sid, sname → sname
sname → sname
} → Trivial FD.

Every Trivial Dependency is always implied in the reln.

Non-Trivial FD:-

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

All possible non-trivial FD:-

× A → B × B → A × C → A
✓ A → C ✓ B → C × C → B
× A → BC × B → AC × C → AB
✓ AB → C
× BC → A
× AC → B

~~Non-Trivial FD:-~~
~~A → C~~
~~B → C~~
~~B → A~~
~~C → A~~
~~A → B~~
~~B → A~~

A	B
1	2
2	1

A → B Non-trivial FD

A → A
B → B
AB → AB
AB → A
AB → B
} → Trivial FD

Properties Of FD's:-

(1) Reflexive FD:- if $X \supseteq Y$ then $X \rightarrow Y$ is reflexive (Trivial)
(2) Transitivity:- if $X \rightarrow Y$ & $Y \rightarrow Z$ then $X \rightarrow Z$
(3) Augmentation:- if $X \rightarrow Y$ then $XZ \rightarrow YZ$. (by splitting rule:-)
(4) Splitting rule:- if $X \rightarrow YZ$ then $X \rightarrow Y$ & $X \rightarrow Z$

e.g.

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

$X \rightarrow Y$

if we add something to closure of X, then its closure will only increase & not decrease, ∴ $XZ \rightarrow Y$, but $XZ \rightarrow Z$ (trivial)

Attribute Closer (X^+) :-

Set of attributes determined by X

$R(ABCD)$

$\{A\to B, B\to C, C\to D\}$

$(A)^+ = \{A,B,C,D\} \Rightarrow A\to \underline{ABCD}$

$A\to A$ (Trivial)

using splitting rule:-

$A\to A, A\to B, A\to C, A\to D$

$(C)^+ = \{C, D)$

means $C\to C$ means $C\to D$

$AB\to CD, AF\to D, DE\to F, C\to G, F\to E, G\to A\}$

which option is false?

a) $(CF)^+ = ACDEFG$ $\longrightarrow ((CF^+)^+ = \{\cancel{G}\, C, F, G, E, A, D\}$

b) $(BG)^+ = ABCDG$ $\longrightarrow (BG)^+ = \{B, G, A, C, D\}$

c) $(AF)^+ = ACDEFG$ $\longrightarrow (AF)^+ = \{A, F, E, D\}$

d) $(AB)^+ = ACDFG$ $\longrightarrow (AB)^+ = \{A, B, C, D, G\}$

superkey :- Let R be the relational schema, & X be ~~the~~ some set of attributes over R, if X^+ (closure of X) determines all attributes of R then X is said to be superkey of 'R'.

$(X)^+ = \{$All attributes of R$\}$

superkey

) $R(ABC)$

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

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

super key $A\to A$ $A\to B$ $B\to C$ super key. super key

Candidate Key (minimal superkey):-

if (X is superkey of R & no ~~supers~~ proper set of X is superkey) then X is the candidate key.

(AB) : Superkey

$A^+ = \{$Not all attributes.$\}$

$B^+ = \{$Not all attributes.$\}$

then (AB) : candidate key.

if superkey with one attribute is always a candidate key.

1* R(ABCDE)
$\{AB \rightarrow C, B \rightarrow E, C \rightarrow D\}$ FD
$(AB)^+ = \{A, B, C, D, E\}$
↓
super key

now, checking whether AB is candidate key or not.
$(A)^+ = \{A\}$ → not super key
$(B)^+ = \{B, E\}$ → not super key
∴ proper subsets of AB are not superkeys, ∴ AB is candidate ke

*if we are not able to determine a good subset of R(ABCDE) for superkey:-

take all attributes:-
• $(ABCDE)^+ = \{A, B, C, D, E)$ → super keys
but $C \rightarrow D$
∴ D is not req. on LHS
∵ $(ABCE)^+ = \{A, B, C, D, E\}$
but $B \rightarrow E$ ∴ E is not req. on LHS
∴ $(ABC)^+ = \{A, B, C, D, E\}$
$AB \rightarrow C$, ∴ C not req. on LHS
∴ $(AB)^+ = \{A, B, C, D, E\}$
now check ~~for its~~ whether it is candidate key.

Q. R(ABCDE)
$\{AB \rightarrow C, C \rightarrow D, B \rightarrow E, E \rightarrow A\}$
$(AB)^+ = \{A, B, C, D, E\}$ → super key
$(A)^+ = \{A\}$ → not super key
$(B^+) = \{B, E, A, C, D\}$ → super key
∴ proper subset of $(AB)^+$ is a superkey, ∴ it is not candidate key.

* if Non-trivial FD
X → Prime Attributes in R, one of the attributes of the primary key
then R consist more than one candidate key.
e.g.

? R(ABCD)

$\{AB \rightarrow CD, D \rightarrow A\}$

$(AB)^+ = \{A, B, C, D\}$ → super key (also candidate key)

$(A^+) = \{A\}$

$(B)^+ = \{B\}$

& ~~there is no X~~

so A,B are prime attributes,

$D \rightarrow A$

so replace A by D in $(AB)^+$

$(DB)^+ = \{D, B, A, C, D\}$ → super key

$(D)^+ = \{D, A\}$ → not super key

$(B)^+ = \{B\}$ → not super key

∴ (D,B) is candidate key.

? R(ABCD)

$\{AB \rightarrow CD, C \rightarrow A, D \rightarrow B\}$

$(AB)^+ = \{A, B, C, D\}$ → super key

$(A)^+ = \{A\}$ → not super key

$(B)^+ = \{B\}$ → not super key

∴ $\boxed{AB}$ is candidate key.

$C \rightarrow A$

∴ replace A by C in AB

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

$C^+ = \{C, A\}$

$B^+ = \{B\}$

∴ $\boxed{CB}$ is candidate key.

$D \rightarrow B$

∴ replace B by D in AB

$(DB)^+ = \{D, B\}$ → not super key.

now, X → prime attribute

$D \rightarrow B$, ∴ replace B by D

∴ $(CD)^+ = \{C, D, A, B\}$

$C^+ = \{C, A\}$

$D^+ = \{D, B\}$

∴ $\boxed{CD}$ is candidate key

$C \rightarrow A$, ∴ replace C by A

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

$A^+ = \{A\}$ ∴ $\boxed{AD}$ is candidate key.

$D^+ = \{D, B\}$

Q. R(ABCDEF)

$\{AB \to C, C \to D, D \to E, E \to F, F \to A\}$

$\boxed{(AB)^+} = \{A, B, C, D, E, F\}$ → super key & candidate key

$A^+ = \{A\}$

$B^+ = \{B\}$

$F \to A$, ∴ replace A by F

$\boxed{(FB)^+} = \{F, B, A, C, D, E\}$ → super key & candidate key

$F^+ = \{F, A\}$

★ $B^+ = \{B\}$

$E \to F$, ∴ replace F by E

$\boxed{(EB)^+} = \{E, B, F, A, C, D\}$ → super key & candidate key.

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

$(B)^+ = \{B\}$

$D \to E$, ∴ replace E by D

$\boxed{(DB)^+} = \{D, B, E, F, A, C\}$ → super key & candidate key

$D^+ = \{D, E, F, A\}$

$B^+ = \{B\}$

$C \to D$, ∴ replace D by C

$\boxed{(CB)^+} = \{C, B, D, E, F, A\}$ → super key & candidate key

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

$B^+ = \{B\}$

$AB \to C$

∴ replace C by AB

$(AB, B) = (A, B)$ which is repeated

Q. R(ABCDEF)

$\{AB \to C, C \to D, D \to E, E \to BF, F \to A\}$

$\boxed{(AB)^+} = \{A, B, C, D, E, F\}$ → super key & candidate key.

$A^+ = \{A\}$

$B^+ = \{B\}$

★ $E \to BF \Rightarrow E \to B$ (by splitting) ∴ replace B by E

$(\cancel{B}AE)^+ = \{A, E, B, F, C, D\}$ → super key & not candidate key

$A^+ = \{A\}$

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

$F \to A$, ∴ replace A by F in AB

$(FB)^+ = \{F, B, A, C, D, E\}$ → super key & candidate key

$F^+ = \{F, A\}$

$B^+ = \{B\}$

$E \to BF$

$\boxed{E^+} = \{A, B, C, D, E, F\}$ → candidate key

$D \to E^+$

$\boxed{D^+} = \{A, B, C, D, E, F\}$ → candidate key

$C \to D$

∴ $\boxed{C^+} = \{A, B, C, D, E, F\}$ → candidate key

?. R(ABC) with no non trivial FD's.

∴ $(ABC)^+ = \{A, B, C\}$

$A^+ = A$ $\quad C^+ = C \quad BC^+ = BC$

$B^+ = B \quad AB^+ = AB \quad AC^+ = AC$

∴ ABC → candidate key.

★ If there is no non-trivial FD, then all the attributes taken together makes the candidate key.

?. R(ABCDE)

$\{A \to B, B \to C, C \to D, D \to A\}$

$(AE)^+ = \{A, E, B, C, D\}$ → superkey & candidate key.

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

$E^+ = \{E\}$

$A \to B$

∴ (BE), (CE), (DE) → candidate keys also.

Q R(ABCDEH)

$\{A \to B, BC \to D, E \to C, D \to A\}$

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

$(ABEH)^+ = \{A, E, B, C, D\}$ → super key

$A^+ = \{A, B\}$

$B^+ = \{B\}$

$E^+ = \{E, C\}$

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

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

$\boxed{(BEH)^+} = \{B, C, E, A, D, H\}$ → super key & candidate key.

$B^+ = \{B\}$

$E^+ = \{E, C\}$

$A \to B$, ∴ replace B by A

$\boxed{(AEH)^+} = \{A, E, B, C, D, H\}$ → super key & candidate key.

$A^+ = \{A, B\}$

$E^+ = \{E, C\}$

$D \to A$, ∴ replace A by D

$\boxed{(DEH)^+} = \{D, E, A, C, B, H\}$ → super key & candidate key

$D^+ = \{D, A, B\}$

$E^+ = \{E, C\}$

$BC \to D$, ∴ replace D by BC

$(BCEH)^+ = \{ABCDEH\}$ → super key

$(BEH)^+ = \{A, B, C, D, E, H\}$ → super key.

∴ BCEH is not a candidate key.

Membership Test:-

$F = \{ . \ldots \} \models X \to Y$ (implication) :→ If F be the FD set & $X \to Y$ be any FD. $X \to Y$ is implied in F only if closure of X (X^+) determin Y.

e.g

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

check if it implies, i.e. $F \models A \to C$

$A^+ = \{A, B, C\}$ ∴ $A \to C$

if $F \models AB \to C$

$(AB)^+ = (A, B, C)$ ↳ $AB \to C$

∴ $AB \to C$

Q. $F = \{AB \to C, C \to D\} \models B \to D$

$B^+ = \{B\} \to$

∴ $B \not\to D$

Q. $F = \{AB \to C, BC \to D\} \models AB \to D$

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

$\boxed{AB \to D}$

Q. Equality of FD sets:-

$F = \{\ \}$

$G = \{\ \}$

F equals to G only if

(i) F covers G:-

All G functional dependencies should be implied in F.

(ii) G covers F:-

All F FD's should be implied in G.

$\{A \to BC, B \to C, AC \to B\} \to F$

$= \{AB \to C, A \to B, A \to C\} \to G$

(i) if F covers G:-

check FD's of F:-

$(AB^+) = \{A, B, C\}$ ∴ $AB \to C$

$(A)^+ = \{A, B, C\}$ ∴ $A \to B$, $A \to C$. F covers G

(ii) check if G covers F:-

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

$A \to BC$

$(B^+) = \{B, \cancel{C}\}\ \{B\}$

$\cancel{B \to C}$ ∴ $B \nrightarrow C$

$(AC)^+ = (A, C, B)$

$AC \to B$

G doesn't cover F

G is subset of F,

∴ $\boxed{G \subset F}$

2) $F_1 = \{A \to B, AB \to C, D \to AC, D \to E\}$

$F_2 = \{A \to BC, D \to AE\}$

check if F_1 covers F_2

- $A^+ = \{A, B, C\}$

  $A \to BC$

- $D^+ = \{D, A, C, E\}$

  $D \to AE$

∴ F_1 covers F_2

∴ $\boxed{F_1 \equiv F_2}$

check if F_2 covers F_1

- $A^+ = \{A, B, C\}$

  $A \to B$

- $AB^+ = \{A, B, C\}$ ~~AB^+~~

  $AB \to C$

- $D^+ = \{D, A, E, B, C\}$

  $D \to AC$

  $D \to E$

∴ F_2 covers F_1

Q. $R(ABCD)$. $\{AB \to CD, D \to A, C \to B\}$

what are the candidate keys of $R1(BCD)$?

$\boxed{(CD)^+} = (C, D, B) \to$ super key & candidate key

$C^+ = (C, B)$

$D^+ = (D, A)$

$AB \to CD$

replace CD by AB

AB

} → My answer

Sir's Answer:- [1] Find FD set of the sub relation R1

for this take all (proper) subsets of R1.

$B^+ = \{B\}\ (B \to B)$

$C^+ = \{C, B\}\ (C \to C, C \to B)$

$D^+ = \{D, A\}\ (D \to D, D \to A)$ but A is not in R1, ∴ no need to add D

$BC^+ = \{B, C\}\ (BC \to BC)$

$CD^+ = \{C, D, A, B\}\ (\underset{\text{trivial}}{CD \to CD}, \underset{\text{A not in R1}}{CD \to A}, CD \to B)$

$BD^+ = \{B, D, A, C\}\ (BD \to BD, BD \to \overset{R1}{A}, BD \to C)$

R1(BCD)

$\boxed{\begin{array}{l} C \to B \\ CD \to B \\ BD \to C \end{array}}$

↑ non-trivial FD's

∴ candidate keys = $\{CD, BD\}$

Q. R(ABCDEF)

{AB→C, B→D, BC→A, D→EF}

What are the candidate keys of R1(ABCD)?

R1(ABCD) → proper subsets →
$A^+ = \{A\}$
$B^+ = \{B, D, E, F\}$
$C^+ = \{C\}$
$D^+ = \{D, E, F\}$
$AB^+ = \{A, B, C, D, E, F\}$
$BC^+ = \{B, C, A, D, E, F\}$
$CD^+ = \{C, D, E, F\}$
$AC^+ = \{A, C\}$
$AD^+ = \{A, D, E, F\}$
$BD^+ = \{B, D, E, F\}$
$ABC^+ = \{A, B, C, D, E, F\}$
$ACD^+ = \{A, C, D, E, F\}$
$BCD^+ = \{B, C, D, A, E, F\}$
$ABD^+ = \{A, B, D, C, ~~E~~, F\}$

As AB^+ is a super key, so any element added to AB makes that set a super key too, ∴ no need to calculate it
e.g. $ABC^+ = \{A, B, C, D\}$ (without calculation)
∴ ABC → D is a FD.

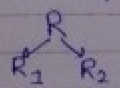
Non-FD of R1
B → D~~EF~~
D → EF ✗
AB → CD~~EF~~
BC → AD~~EF~~
CD → EF
AD → EF
BD → EF
ABC → D
BCD → A
ABD → C

B → D
AB → CD
BC → AD
ABC → D
BCD → A
ABD → C
} → FD of R1(A

Now,
~~R1~~ $(ABCD)^+ = \{A, B, C, D\}$
$(ABD)^+ = \{A, B, C, D\}$
$\boxed{(AB)^+} = \{A, B, C, D\}$ → candidate key
$A^+ = \{A\}$
$B^+ = \{B, D\}$
Now BC → AD
∴ replace ~~B~~ A by BC.
∴ $\boxed{(BC)^+} = \{A, B, C, D\}$ → candidate key
$C^+ = \{C\}$

Properties of Decomposition :-

Lossless Join Decomposition

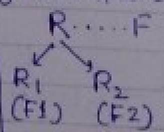


- if $R_1 \bowtie R_2 = R$ (lossless join)
- but if $R_1 \bowtie R_2 \supset R$ (lossy join)

(join b/w subrelations should be equal to original relation)

- $R_1 \bowtie R_2 \subset R$ (not possible)

Dependency preserving Decomposition



F = {X→Y, Y→Z}

$F_1 \cup F_2 \equiv F$ (Dependency preserving)
$F_1 \cup F_2 \subset F$ (not dependency preserving)
$F_1 \cup F_2 \supset F$ (not possible).

Dependency Preserving Decomposition :-

Let R be the Relational schema with FD set F decomposed into R1, R2, …, RN with FD sets F1, F2, …, FN

In general $F1 \cup F2 \cup \ldots Fn \subseteq F$

If $F1 \cup F2 \cup \ldots \cup FN \equiv F$ (Dependency Preserving)

If $F1 \cup F2 \cup \ldots \cup FN \subset F$ (not dependency Preserving)

R(ABCD)

$\{A \to B, B \to C, C \to D, D \to A\}$

$D = \{(AB), (BC), (CD)\}$

) Identify functional dependencies of Subrelations

FD of AB:-

$AB^+ = \{A, B, C, D\}$:⟶ $AB \to CD$

$A^+ = \{A, B, C, D\}$ ⟶ $A \to A$, $\boxed{A \to B}$ ✓, $A \to C, A \to D$

trivial; C, D not in (AB)

$B^+ = \{C, D, A, B\}$ ⟶ $B \to B$, $B \to A, B \to C, B \to D$

FD of BC :-

$B^+ = \{C, D, A, B\}$:- $\boxed{B \to C}$

$C^+ = \{C, D, A, B\}$:- $C \to B$

FD of CD:-

$C^+ = \{C, D, A, B\}$:- $\boxed{C \to D}$

$D^+ = \{D, A, B, C\}$:- $D \to C$, $\boxed{D \to A}$ ⟹ $D \to C$ from CD, $C \to B$ from BC

since $D \to C$ & $C \to B$, ∴ $D \to B$, now $B \to A$ from AB, ∴ $\boxed{D \to A}$

~~we achieved these 3 relns. $A \to B$, $B \to C$, $C \to D$, but we dont have $D \to A$, using transitivity $A \to B$ & $B \to C \Rightarrow A \to C$ & $A \to C$ & $C \to D \Rightarrow A$~~

we have to get $D \to A$ from this approach & not from D^+.

Now, $A \to B$ is in FD of AB, $B \to C$ in FD of BC, $C \to D$ in FD of CD & $D \to A$ in FD of CD, ∴ dependency preserving

. $R(ABCD) \to \{AB \to CD, D \to A\}$

$D = \{BCD, AD\}$

FD's of BCD

$B^+ = \{B\}$

$C^+ = \{C\}$

$D^+ = \{D, A\}$:- $D \to A$ ✗ (because A is not in R(BCD))

$BC^+ = \{B, C\}$

$BD^+ = \{B, D, A, C\}$:- $BD \to C$

$CD^+ = \{C, D, A\}$:- ~~CD → A~~

FD's of AD

$A^+ = \{A\}$

$B^+ = \{B\}$

$D^+ = \{D, A\}$:- $D \to A$

- $D \to A$ is in FD of A
- $AB \to CD$ is not in FD of AD or BCD

$\boxed{F_1 \cup F_2 \subset F}$

ABC:-
$A^+ = \{A\}$
$B^+ = \{B\}$
$C^+ = \{C\}$

correct soln. :- $AB^+ = \{A,B,C\} \Rightarrow \boxed{AB \to C}$
$BC^+ = \{B,C,A\} \Rightarrow \boxed{BC \to A}$
$AC^+ = \{A,C,B\} \Rightarrow \boxed{AC \to B}$

Q. R (ABCDEG)
$\{AB \to C, AC \to B, AD \to E, B \to D, BC \to A, E \to G\}$
$D_1 = \{ABC, ACDE, ADG\}$

FD's of ABC:
$A^+ = \{A\}$ ✓
$B^+ = \{B\}$
$C^+ = \{C\}$
$AB^+ = \{A,B,C\} \Rightarrow \boxed{AB \to C}$
$BC^+ = \{B,C,A\} \Rightarrow \boxed{BC \to A}$
$AC^+ = \{A,C,B\} \Rightarrow \boxed{AC \to B}$

we can't get
$B \to D$ & $E \to G$
from these FD's,
∴ not dependency preserving.

FD's of ACDE
$A^+ = \{A\}$ -
$C^+ = \{C\}$ -
$D^+ = \{D\}$ -
$E^+ = \{E, G\}$ -
$AC^+ = \{A, C\}$
$AD^+ = \{A,D,E\} \Rightarrow \boxed{AD \to E}$
$AE^+ = \{A, E\}$
$CD^+ = \{C, D\}$
$CE^+ = \{C, E\}$
$DE^+ = \{D, E\}$
$ACD^+ = \{A,C,D,E\} \Rightarrow \boxed{ACD \to E}$
$ACE^+ = \{A,C,E\}$
$ADE^+ = \{A,D,E\}$
$CDE^+ = \{C,D,E\}$

FD's of ADG
$A^+ = \{A\}$
$D^+ = \{D\}$
$G^+ = \{G\}$
$AD^+ = \{A, D\}$
$AG^+ = \{A, G\}$
$DG^+ = \{D, G\}$

ans. :- not dependency preserving.

4 : Natural join(⋈)
: projection (π)
: Selection (σ)
: cross product (×)

π- projection:-
π(R)

R

sid	cid	fee
S1	C1	5K
S1	C1	5K
S1	C2	6K

π(R) = cid fee

cid	fee
C1	5K
C2	6K

Result of relational Algebra query always distinct tuples.

σ (selection):-

Selection operator results

$\sigma_p(R)$: results tuples from relation R those are satisfy predicate condition P

"Retrieve sid's who are enrolled cource C2"

$\pi_{sid}(\sigma_{cid = C2})(R)$

Cross Product :-

R × S : results all attributes of R followed by all attributes of S with all combination of tuples R & S.

R

sid	cid	fee
S1	C1	5K
S2	C1	5K
S1	C2	6K

n

S

sid	sname
S1	A
S2	A

m

R × S

sid	cid	fee	sid	sname
S1	C1	5K	S1	A
S1	C1	5K	S2	A
S2	C1	5K	S1	A
S2	C1	5K	S2	A
S1	C2	6K	S1	A
S1	C2	6K	S2	A

m * n

★ If R has X attributes & ~~R~~ S has Y attributes, then R×S has X+Y attributes.

★ If R has 'M' tuples & S has 'zero' tuples, then R×S has zero tuples.

Natural Join :- ($\bowtie$)

$R \bowtie S \Rightarrow$ (1) R×S

(2) $\sigma_{R.Sid=S.Sid}(R \times S)$: Selection of tuples equality b/w common attribute.

(3) ~~π~~ $\pi_{Sid,cid,fee,Sname}(\sigma_{R.Sid=S.Sid}(R \times S))$: projection of distinct columns

$R \bowtie S \rightarrow$

Sid	cid	fee	Sname
S1	C1	5K	A
S2	C1	5K	A
S1	C2	6K	A

Q. R(ABC) S(BCD)

$R \bowtie S = \pi_{A,B,C,D}(\sigma_{R.B=S.B \wedge R.C=S.C}(R \times S))$

Q. R(AB) S(CD)

$\boxed{R \bowtie S = ~~\text{Null}~~ \; R \times S}$ (if no common attributes, then natural join degenerates to cross product.)

Lossless Join Decomposition :-

Let R be the relational schema with decomposed into R1, R2, ..., RN. In general $R_1 \bowtie R_2 \bowtie R_3 \bowtie \ldots \bowtie RN \supseteq R$

if $R_1 \bowtie R_2 \bowtie \ldots \bowtie RN = R$, then it is lossless join decomposition.

if $R_1 \bowtie R_2 \bowtie \ldots \bowtie RN \supset R$, then it is lossy join decomposition

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

decomposed into

R1

A	B
1	1
2	1
3	2

R2

B	C
1	2
1	1
2	2

$R1 \bowtie R2$

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

→ Extra Tuples

$R_1 \bowtie R_2 \supset R$, ∴ lossy join

Decomposed into R1(AB) & R2(AC)

R1

A	B
1	1
2	1
3	2

R2

A	C
1	2
2	1
3	2

$R_1 \bowtie R_2$

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

★ If common attribute $(R_1 \cap R2)$ is superkey of either R_1 or atleast one of reln. then decomposition is lossless.

now $R_1 \bowtie R_2 = R$

∴ lossless join.

★ If common attribute $(R_1 \cap R_2)$ is not super key of ~~both the~~ either R1 or R2 or both, then decomposition is alw[illegible] lossy join.

Requirements → ER Model → Tables → Normalization → Create tables in RDBMS

Q. $R(ABC)$ $\{A \to B, A \to C\}$

~~R~~ $D_1 = \{AB, BC\}$

FD's of AB

$A^+ = \{A, B\} : A \to B$

$B^+ = \{B\}$

FD's of B C

$B^+ =$ ~~{A, B, C}~~ ~~A→C~~ $\{B\}$

$C^+ = \{C\}$

in $R(ABC)$:- $A \to B$ is obtained from FD of AB

$A \to C$ " not " ~~AC~~

lossless join. ∴ ~~lossy join~~ ∴ not dependency preserving.

→ My Answer

Sir's Answer :-

$(R_1 \cap R_2) = B^+ = B$:- not superkey, ∴ Lossy Join.

$D_2 = \{AB, AC\}$

$R_1 \cap R_2 = A^+ = ABC$:- superkey, ∴ lossless join.

?. $R(ABCDE)$

$\{AB \to C, C \to D, B \to E\}$

1. $D = \{ABC, CD\}$

$ABC \cap CD = C$

$C^+ = \{C, D\}$ → not superkey

∴ lossy join. → $(R_1 \cup R_2 \neq R)$ as E is not in any of the decomposition.

2. $D = \{ABC, DE\}$

$ABC \cap DE = \phi$

∴ lossy join

3. $D = \{ABC, CDE\}$

$ABC \cap CDE = C$

$C^+ = \{C, D\}$ → not super key

∴ lossy join

4. $D = \{ABCD, BE\}$

$ABCD \cap BE = B$

$B^+ = \{B, E\}$ → ~~not~~ superkey

∴ ~~lossy join~~ lossless join

★ Let R be the relational schema with FD set F decomposed into R1 & R2, there is lossless join decomposition only if :-

[1] $R1 \cup R2 = R$

[2] $R1 \cap R2 \neq \phi$

[3] $R1 \cap R2 \to R1$ ($R1 \cap R2$ is super key of R1)

(or)

$R1 \cap R2 \to R2$ ($R1 \cap R2$ is superkey of R2)

?. $R(ABCDEG)$

$\{AB \to C, AC \to B, AD \to E, B \to D, BC \to A, E \to G\}$

$D_1 = \{AB, BC, ABDE, EG\}$

(i) union is ABCDEG

(ii) $AB \cap BC = B$

$B \cap ABDE = B$

$B \cap EG = \phi$

∴ lossy join

$D_2 = \{ABC, ACDE, ADG)$

(i) union is ABCDEG

(ii) $ABC \cap ACDE = AC$

$AC \cap ADG = A$

(iii) $A^+ = \{A\}$ → not superkey.

∴ lossy join.

→ My answ

↑ wrong

Sir's answer:-

$D_1 = \{AB, BC, ABDE, EG\}$ (i)st condition is met

ii) $AB \cap BC = B$

$B^+ = \{BD\} \rightarrow$ not superkey

∴ AB & BC can't be join.

$AB \cap ABDE = AB$

$AB^+ = ABCDEG$ (of ABDE)

∴ AB & ABDE can be join.

$AB \cup ABDE = ABDE$

∴ (ABDE) (BC) (EG) are left

$ABDE \cap BC = B$

$B^+ = B \rightarrow$ ~~aa~~ not superkey of either of ABDE & BC

$ABDE \cap EG = E$

$E^+ = \{EG\} \rightarrow$ superkey of EG

∴ (ABDEG) (BC)

$ABDEG \cap BC = B$

$B^+ = \{B\} \rightarrow$ not superkey of either of them → after this, since we get a join problem, we can start again from beginning, ∴ AB join with EG, but $AB \cap EG = \phi$, ∴ we could only start with AB & ABDE

∴ they can't be join.

∴ lossy join

$D_2 = \{ABC, ACDE, ADG\}$ → lossless join.

→ $ABC \cup ACDE \cup ADG = ABCDEG$

$ABC \cap ACDE = AC$

$AC^+ = \{A, C, B, C$

Ans.:- $ABC \cap ACDE = AC$

$AC^+ = \{A, C, B\} \rightarrow$ superkey of ABC

∴ we can join them.

∴ $ABC \cup ACDE = ABCDE$

$ABCDE \cap ADG = AD$

$AD^+ = \{A, D, E, G\} \Rightarrow AD \rightarrow E$ & $E \rightarrow G$, ∴ from transitivity $AD \rightarrow G$

∴ $AD^+ = \{A, D, G\} \rightarrow$ superkey of ADG

∴ we can join them & hence $\boxed{\text{lossless}}$.

~~Since AD→E we can't use E→G & make it AD+ = {A, D, E, G}~~

~~because ADG don't have E in it.~~

~~∴ we can't join them, start again~~

~~ABC ∩ ADG = A~~

~~A+ = {A} → not a superkey for any of them~~

~~∴ we can't join them.~~

~~∴ we start again~~

$ACDE \cap ADG = AD$

$AD^+ = \{A, D, E$

Date

26.08.12

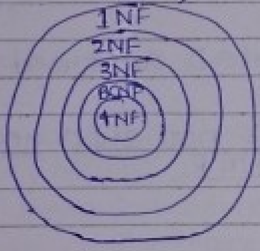
Normal Forms (Eliminate or reduce the redundancy):-

$\underbrace{\text{1. 1NF} \quad \text{2 2NF} \quad \text{3. 3NF} \quad \text{4. BCNF}}_{\text{Single-Valued Function Dependencies } (X \to Y)}$, $\underbrace{\text{5. 4NF}}_{\text{Multivalued FD } (X \to\to Y)}$

i.e. if $X_1 \to Y_1$, then anytime X_1 comes LHS, then RHS will always be Y_1

- Upto BCNF, it eliminates redundancy because of FD. (0% redundancy)
- BCNF relation still suffers from redundancy because of Multivalued Dependency.

★ If relation is in 2NF, then it is in 1NF,
if " " " 3NF, " " " " 2NF, & so on.



First Normal Form :-

Relation R is in 1NF, only if no multivalued attributes exist in R
(R should consist only single valued attribute.

sid	Pno.
S_1	P_1, P_2
S_2	P_3, P_4

Phone no (Multivalued attributes)

Requirement :- ER Diagram (Diagramatic representation of DB) ⟶ Relations (Normalisation) ⟶ RDBMS Table

ER Diagram: Multivalued Attributes allowed

RDBMS Table: Multivalued attributes are not allowed.

★ The by default Normal form of RDBMS is $\boxed{\text{1NF.}}$

Eid	Ename	Pno
E_1	A	P_1, P_2
E_2	B	P_3, P_4
E_3	B	P_3

Pno → Policy No. (Multivalued Attribute)

Not in 1NF (or RDBMS Table)
(Eid→Ename)

⟹

Eid	Pno	Ename
E_1	P_1	A
E_1	P_2	A
E_2	P_2	B
E_2	P_3	B
E_3	P_3	B

(Eid Pno): Candidate key
[include the multivalued attribute into candidate key
(Eid→Ename)

★ R(ABCD) {A→B, B→C}

D is not in FD's, so D is multivalued attribute. (so Reln. is not in 1NF.)
∴ $A^+ = \{A, B, C\}$ → doesn't include D.
∴ include D in candidate key A.
$(AD)^+ = \{A, B, C, D\}$ → multivalued attribute D converted into single valued attribute.

★ if X→Y Non Trivial FD in R with X is not super key.
Rule1:- Then X→Y forms redundancy in R.

X	Y	...
X_1	Y_1	
X_1	Y_1	
X_1	Y_2	
X_2	Y_2	

redundancy

X→Y
not superkey

Rule are imp

★ X→Y non trivial FD with X: super key then (X→Y) doesn't
RULE 2:- cause redundancy.

Q. R(ABCDEF)
{AB→CD, D→A, C→E, D→F}
$AB^+ = \{A, B, C, D, E, F\}$ → Super key
AB→C
AB→D
using Rule1 & Rule 2
AB→CD (no redundancy)
super key

not superkeys
C→E, D→A, D→F
~~no~~ redundancy

candidate key: {AB, DB}

Possible Non-Trivial FD(X→Y) which forms Redundancy.

(i) Proper subset of candidate key → Non prime attribute
Case1 (Always cause redundancy).

from previous ques
only D→F comes under this case

(ii) Non-prime attribute → Non prime attribute
Case 2

(iii) Proper Subset of candidate key → Proper subset of other candidate key.
Case 3

4th Case

(iv) Non-prime attribute → Proper subset of CK. X

R(ABCD)

$\{AB \to C, C \to D, C \to A\}$

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

C→A, ~~repla~~ follows case 4

but we can replace A by C.

$CB^+ = \{B,C,A,D\}$

~~pr~~ super key

∴ C → A

prime attribute (Proper subset of CK). proper subset of other CK

∴ 4th case not possible (& hence no redundancy).

☆ In 1st Normal form, Case1, Case2, Case3 are allowed, i.e. all possible redundancies are allowed.

	Case1	Case2	Case3
1 NF	✓	✓	✓
2 NF	X	~~X~~ ✓	✓
3 NF	X	X	✓
BCNF	X	X	X

✓ → Allowed
X → not allowed

Redundancy Level

1 NF > 2 NF > 3NF > BCNF

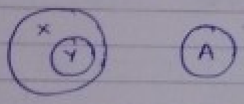
0% Redundancy (Functional Dependency)

Redundancy possible due to multivalued functional dependency.

Second Normal Form:-

Relational Schema R is in 2NF only if
(1) R should be in 1 NF.
(2) R shouldn't consist any partial dependencies (Case 1).

Partial Dependency:-



X: Any candidate key.
Y: Proper subset of candidate key.
A: Non-prime attribute.

Y→A : partial dependency.

Third Normal Form:-

Relational Schema R is in 3NF only if Every non-trivial FD X→Y with
(i) X : Super key
(or)
(ii) Y: prime attribute.

[We can't say that X should be a prime attribute, because in that case, the case(i) may occur.]

$$\underset{\text{super key}}{X} \longrightarrow \underset{\text{prime attribute}}{Y}$$

Case 1:- $\underset{\text{not super key}}{\text{proper subset of CK}} \longrightarrow$ Non prime
∴ not allowed.

Case 2:- $\underset{\text{not super key}}{\text{Non-prime}} \longrightarrow$ Non-prime
∴ not allowed.

Case 3:- $\underset{\text{not super key}}{\text{Proper subset of CK}} \rightarrow \underset{\text{prime attribute}}{\text{proper subset of other CK}}$
∴ allowed.

BCNF :-

Relational Schema R is in BCNF only if every non-trivial F-D X→Y with X should be a super key.
all the 3 cases are not allowed.

Q. R(ABCDE)
{AB→C, C→D, B→E}

example:-

Eid	Pno	Ename
E1	P1	A
E1	P2	A
E2	P2	B
E2	P3	B
E3	P3	B

(Eid Pno): candidate key

$(\underset{\text{proper subset of CK}}{Eid} \to \underset{\text{non prime attribute}}{Ename}) \Rightarrow$ Partial Dependency ∴ not in 2NF

2NF Decomposition

Eid	Pno
E1	P1
E1	P2
E2	P2
E2	P3
E3	P3

Eid	Ename
E1	A
E2	B
E3	B

Eid→Ename

} lossless join
+
2NF / 3NF / BCNF
+
Dep. preservatio

Q. R(ABCDE)
{AB→C, C→D, B→E}

Ans. All elements are in FD's, ∴ in 1 NF
AB⁺ = {A, B, C, D, E}
∴ (A,B) → Candidate key.
but $\underset{\text{proper Subset of CK}}{B} \to \underset{\text{non-prime attribute}}{E}$ (not in 2NF)

Decompose into 2NF:-

$\boxed{ACD}$ $\boxed{BE}$

C→D B→E (now in 2NF)

non prime, non prime, ck, non-prime attribute

(now in 2NF)

but
no, AB→C
so add B in it

$\boxed{ABCD}$

AB→C

ck, non-prime

∴ in (2NF)

C→D (not in 3NF)

non prime, non prime

ABCD ∩ BE = B
B is the super key of BE
∴ lossless join.

& (AB→C & C→D) from ABCD & (B→E) from BE, ∴ dep. preservation.

Check for 3NF:-

ABCD

AB→C (in 3NF)

super key

C→D (not in 3NF)

not super key, not a prime attribute

BE

B→E (in 3NF)

super key

Decompose into 3NF :-

ABCD

$\boxed{ABC}$ $\boxed{CD}$

AB→C
super key
(in 3NF & in BCNF)

C→D
super key
(in 3NF)
(& in BCNF)

$\boxed{BE}$

B→E (& in BCNF)

super key → ABC ∩ CD = C

C is the super key of CD.
∴ lossless join.

Q R(ABCDEF)

{A → BCDEF, BC → ADEF, D → E, B → F}

→ all elements are in FD's, ∴ 1NF

now CK :-

$A^+ = \{A, B, C, D, E, F\}$ ∴ A is CK

, $BC^+ = \{ADEFBC\}$ ∴ BC is CK.

→ now for 2NF (check)

proper subset of CK → Non-prime att. (not allowed)

- A → BCDEF (∵ in 2NF)

$\underset{CK}{A}$

- BC → ADEF (∵ in 2NF)

$\underset{CK}{BC}$

- D → E (in 2NF)
- B → F (not in 2NF)

B ↑ proper subset of CK

F ↑ non prime attribute

Decompose into 2NF :-

$\boxed{ABCDE}$	$\boxed{BF}$
A → BCDE BC → ADE D → E	B → F

ABCDE ∩ BF = B

which is superkey of BF.

∴ lossless join

A → F not in this case,

but $A^+ = \{A, B, C, D, E\}$ → F comes from B → F

∴ A → F ← A → B (from there) & B → F (from 2nd) ⇓ A → F

BC → F not in this case :-

$BC^+ = \{BCDE\}$ B → F

∴ BC → ADE by augmentation BC → FC

& by splitting BC → F

∴ $\boxed{BC \to ADEF}$

∴ dependency preservation.

now check for 3NF :-

A → BCDE (3NF)

A ↑ super key

BC → ADE (3NF)

BC ↑ not superkey

A ↑ prime attribute

D → E (not in 3NF)

D ↑ not super key

E ↑ not prime attribute

B → F (3NF)

B ↑ super key

A→BCD   D→E   B→F
BC→AD

now we need A→E , A→F
& BC→E , BC→F

Decompose into 3NF

$\boxed{ABCD}$ $\boxed{DE}$ $\boxed{BF}$

A→BCD   D→E   B→F
↑ super key   ↑ super key   ↑ super key
BC→AD

BC⁺ = {B,C,A,D}
∴ BC is super key
(all in 3NF)
& in BCNF too.

∴ Dependency Preserving

$\boxed{A→E}$
as we know
A→BCD, ∴ A→D & D→E ⇒ A→E

$\boxed{A→F}$
A→BCD, ∴ A→B
& B→F ∴ A→F

ABCD ∩ DE = D
D is ~~prime~~ super key of DE.
∴ lossjoin of ABCD & DE

ABCDE ∩ BF = B
which is super key of BF
∴ lossless join of ABCDE & BF

$\boxed{BC→E}$
BC→AD
⇒ BC→D
& D→E
∴ BC→E

~~BCF~~ $\boxed{BC→F}$
~~BC→AD~~
B→F
by augmentation
BC→FC
& by splitting
$\boxed{BC→F}$

Q. R(ABCD)
{AB→C, BC→D}
→ All elements in FD, ∴ 1NF
AB⁺ = {A,B,C,D} , A⁺ = {A}, B⁺ = {B}
∴ AB is CK
→ Now, check for 2NF:-
AB → C (in 2NF)
↑ CK
BC → D (in 2NF).
↑ not proper subset of CK
→ Now check for 3NF:-
AB→C (in 3NF)
↑ super key
BC → D (not in 3NF).
↑ not superkey   ↖ non-prime attribute

Decompose into 3NF:-

$\boxed{ABC}$ $\boxed{BCD}$
AB→C   BC→D   (Non-key → Non-key not here)
↓ super key   ↓ super key
(as AB⁺ = {A,B,C})   (as BC⁺ = {B,C,D})

placed here to preserve the lossless join property. initially these were like $\boxed{A}$ $\boxed{BCD}$

∴ in 3NF (& also in BCNF)
& ABC ∩ BCD = BC (super key of BCD)
∴ lossless
& dep preservation.

Q.

Q. R(ABCDEFGHIJ)

{$AB \to C$, $A \to DE$, $B \to F$, $F \to GH$, $D \to IJ$)

all elements in FD, ∴ 1 NF.

$AB^+ = \{A, B, C, D, E, F, G, H, I, J\}$

$A^+ = \{A, \quad , D, E, I, J\}$

$B^+ = \{\quad B, , , , F, G, H\}$

∴ AB is candidate key.

Check for 2NF :-

- $\underset{\text{CK}}{\underset{\uparrow}{AB}} \to C$ (in 2NF)
- $\underset{\text{proper subset of CK}}{\underset{\uparrow}{A}} \to \underset{\text{non prime attribute}}{DE}$ (not in 2NF).
- $\underset{\text{proper subset of AB}}{\underset{\uparrow}{B}} \to \underset{\text{non-prime attribute}}{F}$ (not in 2NF)
- $\underset{\text{non-prime}}{\underset{\uparrow}{D}} \to \underset{\text{non-prime}}{\underset{\uparrow}{IJ}}$ (in 2NF).
- $\underset{\text{non-prime}}{\underset{\uparrow}{F}} \to \underset{\text{non-prime}}{\underset{\uparrow}{GH}}$ (in 2NF)

2NF Decomposition (follow this rule for 2NF decomposition)

for $A \to DE$
- calculate A^+

$A^+ = \{A, D, E, I, J\}$

∴ $\boxed{ADEIJ}$

for $B \to F$
- calculate B^+

$B^+ = \{B, F, G, H\}$

$\boxed{BFGH}$

C is not in both of them

∴ $\boxed{ABC}$

added B because of BFGH.

added A because of ADEIJ

$\boxed{ABC}$

$AB \to C$

$AB^+ = \{ABC\}$

superkey (in 3NF & BCNF)

ABC

$\boxed{ADEIJ}$

$A^+ = \{ADEIJ\}$

$A \to DE$ (in 3NF)

$D \to IJ$ (not in 3NF)

$\boxed{BFGH}$

$B^+ = \{B, F, G, H\}$

$B \to F$ (in 3NF)

$F \to GH$ (not in 3NF)

[all in 2NF now.]

& lossless

& dep. preservatio

Decompose into 3NF :-

$\boxed{ABC}$

$\boxed{ADE}$ $A \to DE$ (in BCNF & 3NF)

$\boxed{DIJ}$ $D \to IJ$ ↑ candidate key(or super key) (in BCNF & 3NF)

$\boxed{BF}$

$\boxed{FGH}$ $F \to GH$ ↑ candidate key (or super key) (in BCNF & 3NF)

[Non-prime → Non-prime not possible here.]

lossless + dep. preserving.

Q. R(ABDLPT)

$\{B \to PT, T \to L, A \to D\}$

→ all elements in FD, ∴ 1NF

$AB^+ = \{A, B, D, P, T, L\}$

∴ AB is the candidate key.

Now check for 2NF :-

- $\underset{\text{proper subset of CK}}{B} \to \underset{\text{non-prime attributes}}{PT}$ (not in 2NF)
- $\underset{\text{proper subset of CK}}{A} \to \underset{\text{non-prime attribute}}{D}$ (not in 2NF)
- $\underset{\text{non-prime}}{T} \to \underset{\text{non-prime}}{L}$ (in 2NF)

Decompose into 2NF:-

- $B \to PT$

  $B^+ = \{B, P, T, L\}$

  $\boxed{BPTL}$

  $B \to BT$

  $T \to L$

- $A \to D$

  $A^+ = \{A, D\}$

  $\boxed{AD}$

  $A \to D$

∴ dep. preserved.

but lossy join because $AD \cap BPTL = \phi$

- but we can't include B in $\boxed{AD}$ or A in $\boxed{BPTL}$, because it will lead to tables in 2NF.

∴ make another table

added because AD $\boxed{AB}$ added because of BPTL $\boxed{BPTL}$ $\boxed{AD}$

lossless + dep. preserving

Q. R(ABCDE)

$\{A \to BC, CD \to E, B \to D, E \to A)$

→ all elements in FD, ∴ in 1NF.

Chec

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

∴ CK = $\boxed{A}$

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

∴ CK = $\boxed{E}$

$BC^+ = \{B, C, A, D, E\}$

∴ CK = $\boxed{BC}$

$CD^+ = \{A, B, C, D, E\}$

∴ CK = $\boxed{CD}$

prime attribute :- A, B, C, D, E

☆ when all attributes in a relation are prime attributes, then relation is in 3NF.

Check for BCNF :-

only $B \to D$ is not in BCNF

B is not superkey.

∴ not in BCNF.

Check for 3NF :-

$B \to D$

↑ not superkey

prime attribute (of CK CD)

$\{A, E, BC, CD\}$

∴ (in 3NF) & no FD with non-prime → non-prime.

∴ in 3NF.

Decompose into BCNF

B added for BD

$\boxed{ABCE}$ $\boxed{BD}$

$A \to BCE$ (because A is CK of original Reln.)

$E \to ABC$ (because E is CK or original reln.)

$BC \to AE$ (same reason)

∴ CK = {A, E, BC} → now CD is not in any reln., ∴ dep. not preserved

$B \to D$

∴ B is CK (∴ in BCNF)

CK = {B}

(

$CD \to E$ (not in any reln.)

∴ $\boxed{CDE}$

$CD \to E$

CK = {CD} →

Q. R(ABCDEFGH)

{AB→C, AC→B, AD→E, B→D, BC→A, E→G}

→ ~~G & H are~~ F & H are not in FD, ∴ not in 1NF.

now, take CK :-

$AB^+ = \{A, B, C, D, E, G\}$

$AC^+ = \{A, B, C, D, E, G\}$

$BC^+ = \{A, B, C, D, E, G\}$

∴ CK = {ABFH, ACFH, BCFH}

prime attributes :-
ABCFH

Check for 2NF :-

- AB → C (in 2NF)
  - AB: proper subset of CK
  - C: prime attribute
- AC → B (in 2NF)
  - AC: proper subset of CK
  - B: prime attribute
- AD → E (in 2NF)
  - AD: not proper subset of CK
- B → D (not in 2NF)
  - B: proper subset of CK
  - D: non-prime attribute
- BC → A & E → G (in 2NF)

Decomposition in 2NF :-

added because of BD

ABCEFGH	BD
AB → CDEG	B → D
AC → BDEG	CK = {B}
BC → ADEG	
E → G (PD)	

not in 2NF (all not in 2NF)

$AB^+ = \{A, B, C, D, E, G\}$

CK = {ABEFH, ACEFH, BCEFH}

$B^+ = \{B, D\}$

∴ BD

now AD → E is not in any reln.

if we add D in 1st reln.

(E → G) should be removed from 1st reln., after removing it all the relatio FD's comes into 2NF.

added because of 1st reln.

ABCFH	BD	ADEG
AB → C (in 3NF)	B → D (in 3NF)	AD → E (in 3NF)
AC → B (in 3NF)	{B}	E → G
BC → A (in 3NF)		{AD}
{ABFH, ACHF, BCFH}		

Decompose into 3NF :-

ABCFH	BD	ADE	EG
AB→C AC→B BC→A	B→D	AD→E	E→G
{ABFH, ACFH, BCFH}	{B}	{AD}	{E}
(not in BCNF)	in BCNF	in BCNF	in BCNF

Decompose into BCNF :-

ABC	ABFH	BD	ADE	EG
AB→C AC→B BC→A	added because of ABC {ABFH}			
{AB, AC, BC}	(in BCNF)			
(in BCNF)				

Q. R(ABCD)
{AB→CD, D→A}
check for BD in 1NF:-

CK:- $AB^+ = \{ABCD\}$
$CD^+ = \{C, D, A\}$
∴ CK = {AB, BD}

Check for BCNF:-
D→A (not in BCNF)

Check for 3NF:-
AB→CD (in 3NF)
↑ super key
D→A (in 3NF)
↑ prime attribute

Decompose into BCNF:-

ABCD → added because of AD.

AD

~~AB→~~
• BD→C (not trivial FD.) (because BD is CK.)

CK = {D}

} lossles + BCNF + not dependency preserv

(not preserving dependency, AB→CD not preserv

CK = {BD}

$BD^+ = \{B, D, A, C\}$ → D→A (though A not in Relm. BCD, but is in FD's) → AB→CD ⇒ AB→C

Imp:-

Sometimes

- Relations are not possible to decompose to BCNF by preserving the dependency.

DB Design Goals	1NF	2NF	3NF	BCNF
① 0% redundancy	×	×	×	× (for MVD) ✓ (for FD)
② Lossless join decomposition	✓	✓	✓	✓
③ Dependency preserving	✓	✓	✓	× (not possible to ensure dependency preservation always)

★ If Reln. R doesn't consist any non-trivial dependency, then R always in BCNF (means in R(ABC), then CK = {ABC}) BCNF fails when there is atleast one non-trivial function dependency.

Binary Relation

R(AB): Relation with two attributes is always in BCNF.

(1) {A → B} → BCNF $A^+ = \{A,B\}$, A → superkey, ∴ in BCNF.

(2) {B → A} → BCNF $B^+ = \{B,A\}$, B → superkey, ∴ in BCNF

(3) {A → B, B → A} → BCNF $A^+ = \{A,B\}$, $B^+ = \{B,A\}$, A & B → superkeys, ∴ in BCNF.

(4) { } → ∴ CK = {AB} & from above rule, it is in BCNF.

★ Relational Schema R consists only simple candidate key, then R always in 2NF but may be not in 3NF or BCNF.

[as proper subset of CK → non-prime attribute]
↓
as CK are of 1 attribute only, so its proper subset is φ,
↓
So this is not going to happen when CK are simple.

e.g.

R(ABCD) → in 3NF → not in 3NF.

{A → B, B → C, C → D, B → A} → in 2NF

$A^+ = \{ABCD\}$ CK = {A, B}

$B^+ = \{ABCD\}$

$C^+ = \{CD\}$

C → D (not in 3NF)

↑ non-prime ↑ non-prime

★ Relational Schema R consists only prime attributes then R is always in 3NF (may or may not in BCNF).
because
~~pri~~ proper subset of CK → proper subset of other CK
(Still possible).

e.g. R(ABCDEF)
$\{AB \rightarrow C, C \rightarrow D, D \rightarrow E, E \rightarrow F, F \rightarrow A\}$
$AB^+ = \{ABCDEF\}$
$FB^+ = \{FBACDE\}$
$EB^+ = \{EBFACD\}$
$DB^+ = \{ABCDEF\}$
$CB^+ = \{ABCDEF\}$

all are in 3NF, because of 2nd condn. i.e right side is prime attribute, ∴ in 3NF, but from this example left side doesn't have super key.

★ Rem. R is in 3NF & only simple candidate keys in R,
∴ R is in BCNF.
because proper subset of CK → proper subset of other CK
but all CK are simple,
so the above FD is not present, & hence in BCNF.

Minimal Cover (Canonical cover)

$F = \{A \rightarrow B, B \rightarrow C, \underline{A \rightarrow C}, AB \rightarrow A\}$ → trivial dependency

↳ can be derived from $A \rightarrow B$ & $B \rightarrow C$,
∴ redundant FD
(FD which can be derived from other FD's in FD set F)

$F_m = \underbrace{\{A \rightarrow B, B \rightarrow C\}}$
↳ minimal cover of FD set F.

★ If F_m is minimal cover of F, then
$F_M \equiv F$ (always).

★ Minimal cover is the process of identifying redundant FD.

Extraneous Attributes

(1) $\{AB \to C, A \to B\} \equiv$
$\{A \to C, A \to B\} \equiv \{A \to BC\}$

(2) $\{AB \to C, A \to C\} \equiv \{A \to C\}$

(3) $\{A \to BC, \overset{A \to B}{B \to C}\} \equiv \{A \to B, B \to C\}$
$\Downarrow$
$A \to B$ &
$A \to C$ (but it can be formed using $A \to B$ & $B \to C$)

(4) $\{AB \to CD, BC \to D\} \equiv$
$\Downarrow$
$AB \to C$
$AB \to D$
$\{AB \to C, BC \to D\}$
$AB^+ = \{ABCD\}$
$\therefore AB \to D$
(pseudo transitive property)

★ $F_1 = \{AB \to C, A \to B\}$
$F_2 = \{A \to C, A \to B\}$
check F_1 covers F_2
$A^+ = \{A, B, C\}$
$\therefore F_1$ covers F_2
check F_2 covers F_1
$A^+ = \{A, B, C\}$
$AB^+ = \{A, B, C\}$
F_2 covers F_1
$\therefore \boxed{F_1 \equiv F_2}$

★
$\{XYZ \to W, X \to Y, X \to Z\}$
$\{X \to W, X \to Y, X \to Z\}$
$= \{X \to WYZ\} = F_m$

★ $F = \{AB \to C, A \to B, B \to A\}$
F_m ~~$AB \to C$~~
$A \to C, A \to B, B \to A$
$\Downarrow$
$F_{m_1} = \{A \to BC, B \to A\}$

or

$B \to C, A \to B, B \to A$
$\Downarrow$
$B \to AC$
$F_{m_2} = A \to B$

★ Minimal cover may not be unique, but all minimal covers are logically equivalent to each other.

if $\boxed{A \to C}$
then $AB \to BC$
if $AB \to CB$
then $AB \to C$

if $A \to C$
then $AB \to C$
$ABD \to C$
$ABDE \to C$

pseudo transitivity:-
if $(AB \to C, BC \to D)$ then
$AB \to D$

Q. $\{ABCD \to EF, AD \to BC, E \to F, AB \to C\}$
$AD \to BC$
$AD^+ = \{ADBCEF\}$
$\therefore AD \to EF$
$\therefore \{AD \to EF, AD \to BC, E \to F, AB \to C\}$

$AD \to BCEF, E \to F, AB \to C$
but $E \to F$
$\therefore AD \to BCE, E \to F, AB \to C$
$\downarrow$
$AD \to BC$

$AB \to C$
$AB \to C$
$AB^+ = \{A, B, C\}$
~~$AB^+ =$~~

$F_m = \{AD \to BE, E \to F, AB \to C\}$

$B \to A$ $C \to A$
$B \to C$ $C \to B$ ×

$\to AE$?
$CD \to E \to C$
$CD \to C$
$D \to A$

Q. $\{A \to B, B \to AC, C \to AB\}$

$A^+ = \{A, B, C\}$

$F_m = \{A \to B, B \to C, C \to A\}$

Q. $\{A \to BC, CD \to E, E \to C, D \to AEH, \underline{ABH \to BD}, DH \to BC\}$

$\Downarrow$
$A \to B$
$A \to C$

$D \to E$, $E \to C$

$E^+ = \{E, C$

$ABH \to D$ (trivial)
$AH \to D$ $(A \to B)$

$\{A \to BC, CD \to E, E \to C, D \to AEH, AH \to D, DH \to BC\}$

$\Downarrow$
$D \to A$
$D \to E$
$D \to H$

from this
$CD \to ACEH$

$\{A \to BC, E \to C, D \to AEH, AH \to D, DH \to BC\}$

$D \to H$

$DH \to B$
$DH \to C$
∴ H not req. here

$\{A \to BC, E \to C, D \to AEH, AH \to D, D \to BC\}$
$= \{A \to BC, E \to C, D \to AEH\cancel{BC}, AH \to D\}$

removed
because $A \to BC$

$= \{A \to BC, E \to C, D \to AEH, AH \to D\}$

Multivalued Dependencies :- (MVD)

$(\twoheadrightarrow)$

Redundancy in Relation R

(Non-trivial FD) FD $(X \to Y)$ ↳ not superkey

MVD $(X \twoheadrightarrow Y)$ ↳ not Superkey (Non-trivial MVD.)

★ R(ABCD)

{A→B C→D}
Two independent reln.
Candidate key:(AC)

} → when independent reln. residing in diff. tables are merged & independent FD's comes into same table, then redundancy occurs.

possible redundancy because MVD:-
If two or more multivalued attributes in R converted into single valued attributes, then R suffers from redundancy because of MVD.

e.g.

Sid	Pno	Cno
S1	P1,P2	C1/C2
S2	P1	C1

→

Sid	Pno	Cno
S1	P1	C1
S1	P1	C2
S1	P2	C1
S1	P2	C2
S2	P1	C1

redundancy

(Add Pno & Cno to candidate key)

* No non-trivial FD(∵ BCNF).
* Not free from redundancy (MVD)

MVD:- Let R be the relational schema & x,y be the attribute sets over R.
Z is R-{XUY} {All attributes of R except X,Y

$X \to\to Y$ exists in R only if t_1,t_2,t_3,t_4 tuples ∈ R
(a) $t_1.x = t_2.x = t_3.x = t_4.x$
& (b) $t_1.y = t_2.y$ and $t_3.y = t_4.y$
& (c) $t_1.z = t_3.z$ and $t_2.z = t_4.z$

MVD rules:-

(1) Complementation Rule:-
if $X \to\to Y$ then $X \to\to R-(X \cup Y)$
[R(ABCD) if $A \to\to B$ then $A \to\to CD$]

(2) Trivial MVD:-
$X \to\to Y$ is trivial only if
$X \supseteq Y$ or $X \cup Y = R$
e.g. A(ABCD) $AB \to\to A$, $AB \to\to AB$
$AB \to\to CD$, $A \to BCD$

$A \to B$ }
$B \to A$ } non trivial
R(AB)
$A \to\to B$ }
$B \to\to A$ } → trivial
$AB \to AB$

(3) Transitivity :-

if $X \rightarrow\rightarrow Y$ & $Y \rightarrow\rightarrow Z$ then $X \rightarrow\rightarrow (Z-Y)$ [All attributes of Z except Y]

e.g. $AB \rightarrow\rightarrow C$, $C \rightarrow\rightarrow DE$ then $AB \rightarrow\rightarrow DE$

$AB \rightarrow\rightarrow CD$, $CD \rightarrow\rightarrow DE$ then $AB \rightarrow\rightarrow E$

$\{D,E\} - \{C,D\} = \{E\}$

(4) Augmentation :-

if $X \rightarrow\rightarrow Y$ & $Z \supseteq W$ then $XZ \rightarrow\rightarrow YW$

if $A \rightarrow\rightarrow B$ then
$AC \rightarrow\rightarrow B$
$ACD \rightarrow\rightarrow BC$
$ACD \rightarrow\rightarrow BCD$
$ACD \rightarrow\rightarrow BD$

(5) Relication :-

Every FD is also MVD

if $X \rightarrow Y$ then $X \rightarrow\rightarrow Y$

(6) MVD not allowed to split

$\{X \rightarrow\rightarrow YZ\} \neq \{X \rightarrow\rightarrow Y, X \rightarrow\rightarrow Z\}$

$\{X \rightarrow YZ\} = \{X \rightarrow Y, X \rightarrow Z\}$

Q. True or not :-

(i) if $X \rightarrow YZ$ then $X \rightarrow\rightarrow Y, X \rightarrow\rightarrow Z$ (T)

(ii) if $X \rightarrow\rightarrow YZ$ then $X \rightarrow\rightarrow Y, X \rightarrow Z$ (F)

(iii) if $X \rightarrow YZ$ then $X \rightarrow Y, X \rightarrow\rightarrow Z$ (T)

(iv) if $X \rightarrow\rightarrow YZ$ then $X \rightarrow Y, X \rightarrow\rightarrow Z$ (F)

sid	pno	cno
S1	P1	C1
S1	P1	C2
S1	P2	C1
S1	P2	C2
S2	P1	C1

$\{\overset{\text{not superkey}}{sid} \rightarrow\rightarrow Pno, \overset{\text{not superkey}}{sid} \rightarrow\rightarrow cno\}$ ($\because$ not in 4NF). but in BCNF.

(Non-trivial MVD)

Decomposition $\xrightarrow{\text{to 4NF}}$

sid	pno
S1	p1
S1	p2
S2	p1

$sid \rightarrow\rightarrow pno$
(sid, pno) :- CK

sid	cno
S1	C1
S1	C2
S2	C1

$sid \rightarrow\rightarrow cno$
(sid, cno) :- CK

4NF
no nontrivi
FD
&
No nontrivial
MVD.

4NF :- Relation R is in 4NF only if :

(1) Every non trivial FD X→Y with X should be a super key (BCNF).

(2) Every non trivial MVD X→→Y with X to be a super key.

Note :- Super key or candidate key is always determined FD's only because single valued functional dependencies are key dependencies. MVD's are data dependencies.

Query Languages :-

procedural Query Languages
→ What to retrieve from DB & how to retrieve from DB.

Relational Algebra
π, σ, ×, ⋈, etc.

non-procedural Query Languages
→ what to retrieve from DB.

Relational Calculus

Tuple Relational Calculus (SQL)
(by IBM)

Domain Relational Calculus
~~(QBE)~~ (QBE)
Query By Example
(by Microsoft)
Almost Exhausted

TRC, SQL, RA

- Query condn. evaluates row by row on data base table with one row at a time.

TRC → uses first order logic & predicate calculus.

First Order logic :- ∧, ∨, ¬, →, ↔

Predicate Quantifiers :- ∃, ∀

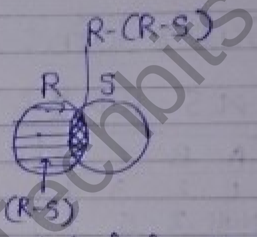
Relational Algebra

Basic Operator:

π :- projection
σ :- selection
X :- cross product
U :- union
— :- set difference
ρ :- rename

Derived Operators:

$\bowtie$: join (π, σ, X)
∩ : intersection $R \cap S = R - (R - S)$
/ : division



R

A	B	C
1	2	3
3	1	2

S

B	C	D
2	3	4
2	5	1

★we need to join the tables because we can't compares R.C & S.C because that means comparison & execution & retrieval of 2 rows at a time which is not possible.

RXS

mxn

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

⇒Conditional Join:- $\bowtie_C$

$$\boxed{R \bowtie_{R.C<S.C} S = \sigma_{R.C<S.C}(R \times S)}$$

★To compare two rows of same table, we need to perform self join of the same table.

Natural Join ($\bowtie$):-

$$R \bowtie S = \pi_{ABCD}(\sigma_{\substack{R.B=S.B \wedge \\ R.C=S.C}}(R \times S))$$

Outer Join :-

(i) Left Outer Join
(ii) Right " "
(iii) Full " "

Left Outer Join:-

R ⟕ S = R ⋈ S & tuples from R that failed join condn.

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

~~Atleast~~
Exact no. of tuples as in R.

Right Outer Join:-

~~R ⟕~~ R ⟖ S = R ⋈ S & tuples from S that failed join condn

A	B	C	D
1	2	3	4
Null	2	5	1

Exact no. of tuples as in S.

Full outer Join:-

R ⟗ S = (R ⟖ S) ∪ (R ⟕ S)

A	B	C	D
1	2	3	4
3	1	2	NULL
NULL	2	5	1

∪, ∩, − (set operations)

Union Compatible:-

R & S are union compatible only if :-

(1) no. of columns of R & S should be same.

(2) Range of attribute of R & S should be similar.

character

sid	sname

sid	marks

→ integer

attributes are of diff. types,
so ∪, ∩, − not possible.

Sid	Sname

~~sid~~ studid	Studname

$\cup, \cap, -$ possible even though names will/be are diff. (only condn. is that their ranges must be similar.)

☆ In that case, result will take column names from 1st relation.

Date

01.09.12

Division:-

E

sid	cid
S1	C1
S1	C2
S2	C1
S2	C2
S1	C3
S3	C1

C

cid
C1
C2
C3

⇒ sid of student enrolled some course.
for this, project distinct sid's ⇒ $\pi_{sid}(E)$

⇒ sid of student enrolled every course. (Division Operator).

$\pi_{sid,cid}(E)/\pi_{cid}(C)$ [This retrieves sid which has all the cids as given by denominator $\pi_{cid}(C)$]

result :- $\boxed{S1}$ since $\boxed{\pi_{cid}(C) = C1, C2, C3}$

Division Operator is derived Operator.

$$\pi_{sid\,cid}(E)/\pi_{cid}(C) = \pi_{sid}(E) - \pi_{sid}(\overbrace{\pi_{sid}(E)\times C}^{\text{every student enrolled every course}} - E) = \begin{bmatrix}S1\\S2\\S3\end{bmatrix} - \begin{bmatrix}S2\\S3\end{bmatrix} = \boxed{S1}$$

Student enrolled every course.

(Select (distinct) sid
from E)
MINUS
(Select distinct sid
from ~~E,C~~ ((Select
*~~distinct sid~~ from
E,C) MINUS
(Select * from
E)));

$$\pi_{sid}(E)\times C \Rightarrow \begin{bmatrix}S1\\S2\\S3\end{bmatrix} \times \begin{bmatrix}C1\\C2\\C3\end{bmatrix}$$

Student enrolled some courses

All courses

⇓

Every student enrolled every course

sid	cid
S1	C1
S1	C2
S1	C3
S2	C1
S2	C2
S2	C3
S3	C1
S3	C2
S3	C3

[Universal Set]

−

sid	cid
S1	C1
S1	C2
S2	C1
S1	C3
S3	C1

Students enrolled some course

=

sid	cid
S2	C3
S3	C2
S3	C3

Students not enrolled all courses or Student enrolled only proper subset of courses.

(A∪B)∩(A∪B)'

Q. suppliers (sid, sname, rating)
parts (pid, pname, colour)
catalog (sid, pid, cost)

(a) Retrieve sid of the suppliers who supplies some red part

Ans. $\pi_{sid}(\sigma_{colour=red}(catalog \bowtie parts))$ → My answer

Catalog

SID	PID	Cost
S1	P1	
S1	P2	
S2	P1	

×

parts

pid	pname	col
P1		G
P2		R
P3		B

=

sid	pid	cost	pid	pname	col
S1	P1		P1		G
S1	P1		P2		R
S1	P1		P3		B
S1	P2		P1		G
S1	P2		P2		R
S1	P2		P3		B
S2	P1		P1		G
S2	P1		P2		R
S2	P1		P3		B

$\pi_{sid}(\sigma_{colour=red \wedge catalog.pid=parts.pid}(catalog \times parts))$

Or

$\pi_{sid}(catalog \bowtie (\sigma_{colour=red}(parts)))$
[More efficient]

(b) Retrieve sid of the supplier who supply some red or some green part.

Ans. $\pi_{sid}(catalog \bowtie (\sigma_{colour=red \vee colour=green}(parts)))$

Or

$\pi_{sid}(catalog \bowtie \sigma_{colour=red}(parts)) \cup$
$\pi_{sid}(catalog \bowtie (\sigma_{colour=green}(parts)))$

(c) Retrieve sid of the supplier who supply some red part & some green part.

$\pi_{sid}(catalog \bowtie (\sigma_{colour=red}(parts))) \cap$
$\pi_{sid}(catalog \bowtie (\sigma_{colour=green}(parts)))$

but not

$\pi_{sid}(catalog \bowtie (\sigma_{colour=red \wedge colour=green}(parts)))$ [This is wrong, because colour will be either green or red but not both.]

or

$\pi_{c1.sid}\left(\sigma_{(c1.pid=p1.pid \wedge p1.colour=red) \wedge (c2.pid=p2.pid \wedge p2.colour=green) \wedge (c1.sid=c2sid)}(c1 \times c2 \times p1 \times p2)\right)$ [correct soln.]

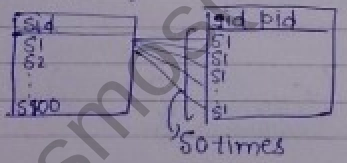
(gondu -- :p)

Q. suppliers(sid, sname, rating) with 100 tuples.
catalog (sid, pid) with 50 tuples.
max. no. of tuples in (suppliers ⋈ catalog).

$\sigma_{suppliers.sid = catalog.sid}(suppliers \times catalog)$

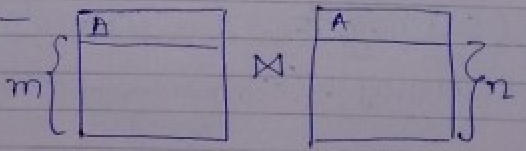
soln. :- Assume sid is the primary key of suppliers, but not of catalog.

at max.,



∴ max. no. of tuples = 50.

Generalisation:-



∴ max. no. of tuples = n

Q. Relation R($\underline{A}$BC), S($\underline{AD}$E), P($\underline{D}$FG) are the 3 reln. with 20,30,40 tuples respec, how many max. no. of tuples in R ⋈ S ⋈ P

Ans. $\boxed{30}$. (or 20??) – [illegible]

Q. Retrieve sid of the supplier who supply every part.

Ans

π_{sid} ~~ks~~ ($\pi_{sid,pid}$~~s~~ (catalog ~~⋈ parts~~) / π_{pid}(parts))

⇓

$\pi_{sid}(catalog) - \pi_{sid}(\pi_{sid}(catalog) \times Parts - catalog)$

Q. Retrieve sid of the supplier who supply every red part.

Ans.

$\pi_{sid\,pid}(catalog) / \pi_{pid}(\sigma_{colour=red}(parts))$

Q. Retrieve sid of the supplier who supply atleast two parts.

Ans. same supplier but diff. parts (if it supplies 1 part, then condn. fails because though $t_1.sid = t_2.sid$, but $t_1.pid = t_2.pid$)

σ (catalog × catalog)

$\pi_{t_1.sid}(\sigma_{t1.sid=t2.sid \wedge t1.pid \neq t2.pid}(\rho(t_1, catalog) \times \rho(t_2, catalog)))$

sid	pid	cost
S1	p1	
S1	p2	
S2	p1	

×

sid	pid	cost
S1	p1	
S1	p2	
S2	p1	

=

	sid	pid	cost	sid	pid	cost
✗	S1	p1		S1	p1	
✓	S1	p1		S1	p2	
✗	S1	p1		S2	p1	
✓	S1	p2		S1	p1	
✗	S1	p2		S1	p2	
✗	S1	p2		S2	p1	
✗	S2	p1		S1	p1	
✗	S2	p1		S1	p2	
✗	S2	p1		S2	p1	

sid of supplier who supply atleast 3 parts :-

$\pi_{t_1.sid}\left(\sigma_{(t_1.sid = t_2.sid = t_3.sid) \wedge (t_1.pid \neq t_2.pid \neq t_3.pid)}\left(\rho(t_1, catalog) \times \rho(t_2, catalog) \times \rho(t_3, catalog)\right)\right)$

not possible, so write it like this

$(t_1.sid = t_2.sid) \wedge (t_1.sid = t_3.sid) \wedge (t_2.sid = t_3.sid)$

Q. sid of the suppliers who supply exactly two parts?

Ans. (sid of suppliers atleast 2 parts) - (sid of suppliers atleast 3 parts)

~~2,3,4~~ — 3,4,
2
3,4 ←

Q. sid of the suppliers who supply atmost two parts?

Ans.

~~sid of suppliers who supply~~

$|\pi_{sid}(\text{suppliers})|$ - |sid of suppliers supplying atleast three parts.|

$\underbrace{|\pi_{sid}(\text{catalog})| - |\text{sid of suppliers supplying atleast three parts.}|}_{\text{sid of suppliers supplying atleast one \& atmost 2 parts.}}$

[because in catalog relm the supplier entry will be done when it has supplied atleast one part.]

Q. Retrieve sid of supplier who has supplied most expensive part

catalog - | supplier who doesn't supply most expensive part |

sid	pid	cost	sid	pid	cost	
s1	p1	10	s1	p1	10	✓
s1	p1	10	s1	p2	20	✓
s1	p1	10	s2	p1	30	×
s1	p2	20	s1	p1	10	✓
s1	p2	20	s1	p2	20	×
s1	p2	20	s2	p1	30	×
s2	p1	30	s1	p1	10	×
s2	p1	30	s1	p2	20	×
s2	p1	30	s2	p1	30	×

$T_1.cost < T_2.cost$

← gives tuples which doesn't have max cost.

$$\pi_{sid}\Big(catalog - \pi_{t_1.sid,\, t_1.pid,\, t_1.cost}\big(\sigma_{t_1.cost < t_2.cost}(\rho(t_1, catalog) \times \rho(t_2, catalog))\big)\Big)$$

Q Retrieve pairs of sid such that supplier with sid1 should charge more than supplier with sid2 for some part. {

$$\pi_{t_1.sid,\, t_2.sid}\Big(\sigma_{(t_1.sid \neq t_2.sid) \wedge (t_1.cost > t_2.cost) \wedge (t_1.pid = t_2.pid)}(\rho(t_1, catalog) \times \rho(t_2, catalog))\Big)$$

Q. Retrieve suppliers who supply 2nd most expensive part.

Q. Retrieve sid who supply atleast two red parts.

$$\pi_{sid}\left(\sigma_{(t_1.sid \neq t_2.sid)}\left(\overset{t_1}{\left(catalog \bowtie \pi_{pid}\left(\sigma_{colour=red}(parts)\right)\right)} \times \overset{t_2}{\left(catalog \bowtie \pi_{pid}\left(\sigma_{colour=red}(parts)\right)\right)}\right)\right)$$

Q Retrieve sid of suppliers who supply least expensive part.

$$catalog - \left|\text{suppliers who doesn't supply least expensive part}\right| = \pi_{sid}(catalog) - \pi_{sid}\left(\sigma_{t_1.cost > t_2.cost) \wedge (t_1.sid \neq t_2.sid)}\left(\rho(t_1, catalog) \times \rho(t_2, catalog)\right)\right)$$

Renaming Columns

Catalog $\quad \rho_{s,p,c}(catalog)$

sid	pid	cost

s	p	c

$$\pi_{sid}\left(Catalog \bowtie_{sid=s \wedge pid \neq p} \rho_{s,p,c}(catalog)\right)$$

[supplier supplying atleast 2 parts.]

since column names are diff, hence it degenerates to cross product.

Page no. 77

Q23.

$$\underbrace{\pi_{name}(\sigma_{sex=Female}(student))}_{\text{Names of all Female}} - \pi_{name}(student \bowtie_{\substack{sex=female \\ \wedge\, x=male \\ \wedge\, marks<=m}} \rho_{n,x,m}(student))$$

Name	Sex	marks
A	M	10
B	F	20
C	M	30
D	F	40
E	M	50
F	F	60

it gives names of female students who score less marks than some male students

$$\pi_{name}((\sigma_{sex=female}(student)) \bowtie \sigma_{\substack{marks \\ <=m}}(\rho_{n,x,m}(student)))$$

name	marks
B	20
D	40
F	60

n	m
A	10
C	30
E	50

⇓

name			
B	20	C	30
B	20	E	50
D	40	E	50

★ Whenever there is a natural join or cross product, ~~th~~ it gives 'some' & not 'all'.

★ for 'all', we use division operator.

so query gives:-
names of female students who score more marks than 'all' male students.
(Minus gives the compliment).

SQL :- (Structured Query Language)

(1) DDL (Data Definition Language)
To modify structure of DB table.
e.g. Create Table, Drop Table, Alter Table Add/remove Attributes

(2) DML (Data Manipulation Language)
To modify database records (data)
e.g. Insert into, Delete from, Update Set

(3) DCL (Data Control Language) [Transaction & Consistency control]
Transaction based, commit, rollback (Abort).

(4) DQL (Data Query Language)
(Retrieve data from DB)
e.g. Select, Group By, where, having

$$\overset{\text{projection}}{\overbrace{\text{Select Distinct } A1,A2,\ldots,AN}} \text{ From } \overset{\text{cross product}}{\overbrace{R1,R2,\ldots,RM}} \text{ where } \underset{\text{cond}^n\text{ of selection operator}}{P}$$

$$\Downarrow$$

$$\pi_{A1,A2,\ldots,AN}(\sigma_P(R1 \times R2 \times \ldots \times RM))$$

Q. Retrieve sid of the supplier who supply some red part.

Ans. • Select distinct sid from parts, catalog where catalog.pid = parts.pid AND catalog.colour = 'RED';

$$\boxed{\pi \equiv \text{Select distinct}}$$

Basic SQL Clauses :-

Select [distinct] A1,A2,....,AN from R1,R2,....,RM
[where P] [Group by (Attributes)[Having condition]]
[order by attributes [DESC]]

① From Clause :- Cross product (×)
② Where clause :- Selection Operator (σ)
③ Group By (Attribute)

sid	Marks	branch
S1	40	CS
S2	50	ES
S3	80	CS
S4	80	IT
S5	40	IT
S6	NULL	EC

Group By(branch)

S1	40	CS	2
S3	80	CS	
S2	50	EC	2
S6	NULL	EC	
S4	80	IT	2
S5	40	IT	

Aggregate Operators

① COUNT ([DISTINCT] Attributes)
② SUM (" ")
③ AVG (" ").
④ MIN (Attributes)
⑤ MAX (")

→ NULL values discarded by aggregation.
→ Arithmetic operations with NULL results NULL.
e.g. NULL+5 = NULL.

★ Count(*) = no. of records/tuples → ⑥
★ count(marks) = no. of non-null marks → ⑤
★ count(distinct marks) = ③
★ sum(marks) = sum of non-null marks = 290
★ sum(distinct marks) = 170
★ AVG(marks) = SUM(marks)/COUNT(marks)
★ AVG(distinct marks) = SUM(distinct marks)/COUNT(distinct marks)

Group by Clause is not possible to derive using basic relational algebra.

Q.

R

A	B	
a1	~~10~~	15
a2	~~20~~	25
a3	~~30~~	35
a4	~~NULL~~	NULL.

- Update R set B = B+5.
- Select Avg(B) from R

$$\text{Avg}(B) = \frac{\text{Sum}(B)}{\text{Count}(B)} = \frac{75}{3} = 25$$

Q. Select $\underbrace{\text{min}(m)}_{\text{gives min. marks}}$, sid from Stud. → not valid

[Aggregation fn. is not allowed along with other attribute in select clause.]

min	branch
40	CS
50	EC
40	IT

↑

This means that only those attributes are allowed in group by clause along with the aggregate fn. which are there in the group by clause.

- select min(m),branch from stud group by branch. [this is allowed]

★ alongwith the aggregate function allowed to select other attribute in select clause only if other attribute is in Group by clause.

★ If Group By clause exist, aggregate fn. in group by clause is applied for every group.

Q. select ~~min(marks)~~ min(max(marks)) from stud group by branch. (?)

O/P:-

80
50
80

the group by is used for both max. & min. aggregate fn. & not only for max(marks).

∴ Nested Aggregation is not useful in the SQL.

∴ $\boxed{\text{Avg}(\min(\max(m))) = \max(m)}$

④ Having Clause:-

[where is applied for every record, having is " " each group.]

Selection of groups that satisfy 'having' condition.

★ Select students whose branch Avg. Marks greater than 50.

Select ~~stud~~name from ~~stud~~ where

Select sid, branch from stud group by branch having Avg(Marks)>50;

[★If the select clause doesn't have aggregate fn.

We can use having clause without group by clause (If group by clause doesn't exist, having clause condn. is applied to each record & hence having clause = where clause).

select sid,branch from stud S1 where marks> (select Avg(marks)
from studS2 where
S1.branch=S2.branch)
group by sid,branch;

⑤ Select }
⑥ Distinct } → equivalent to (π) [select distinct = π]

⑦ Order By:-

It is meant for Ascending or Descending ordering of records.

sid	sname
S1	C
S2	A
S3	B

select sid,sname from stud order by sname

this attribute must be in the select attribute.

Set Operations:-

- Union / Union All
- Intersect / Intersect All
- Minus (Except) / Minus All

Reln. should be union compatible.

·Union, Intersect, Minus → result is distinct records.

R
2
2
2
3
3
4

S
2
2
3
5
5

R Union S

R

A
2
3
4
5

R Union All S

A
2
2
2
3
3
4
4
2
2
3
5
5

R intersect S

A
2
3

R intersect ALL S

A
2
2
3

R Minus S

A
4

R Minus ALL S

2
3
4
4

- Select sid from stud where marks= max(marks),
  we can't do this because where clause is applied tuple by tuple & results in all tuples of the reln.
  because max(marks) for one tuple is that marks itself;
  ∴ marks=marks & hence all tuples are selected.

Correction:-

Select sid from stud where marks= (select max(marks) from stud);

Nested Queries :-

Query inside Query

Independent Nested Queries

(Inner Query is independent of outer query.)

→ Bottom-Top

(Inner query is executed first.)

→ Best Used Operators

IN, ANY, ALL

can be used for correlated nested queries.

Correlated Nested Query

(Inner Query uses attribute specified in the outer query.)

→ ⑥ select R.A
① From R
② where ... (Select S.B ⑤
From S ③
where S.B = R.A) ④

→ Top-Bottom-Top

→ Best Used Operators

Exists

can be used for independent nested queries.

IN Operator:-

To check given tuple is member of set of tuples or not:

X IN {2, 3, 5, 7, 10}

if X=5 IN returns True

if X=6, IN returns False

e.g. sid of the suppliers supplying RED part.

```
Select sid
from Catalog
where pid IN(select pid from
             from parts
             where colour = 'RED')
```

★

sid	pid
s1	p1
s2	p2
s3	p3

pid	colour
P1	R
P2	G
P3	R

→ first inner query is executed which returns {P1, P3}

→ then outer query is executed which gives {P_1, P_3} as o/p.

Or

Select sid from catalog C, parts p where c.pid = p.pid and p.colour = 'RED'; [this is less efficient than nested subqueries because of cross product.]

★ NOT IN is complement of IN

ANY Operator {∃: 'There exist'} or some, any, atleast one.

- Operators that can be used by 'ANY' operator: <, <=, >, >=, =, <>

A1 operator ANY{. . . }

e.g.

X < ANY{2,3,4,5,7,10}

ANY returns true only if:-

Atleast one tuple in subquery result should satisfy comparis operation of given value.

e.g. X = 4 ANY returns true

∃sid (sid scored > 95%)

& X < ANY{empty set} → returns false (If inner query results empty, any always returns false.)

IN Equal to

IN is equivalent to '= ANY'.

★If inner query result is empty, IN operator results false.

ALL Operators:- {∀ in all (Every)

A1 operator ALL{ }

x < ALL{2,3,5,7,10}

if x=0 ALL returns true

if x=6 ALL returns false.

ALL operator returns false only if atleast one tuple in the inner query should fail the comparison operator.

x < ALL{Empty}

ALL returns false only if atleast 1 tuple failed the condition.

ALL operator returns true if inner query result is empty.

$\boxed{\{\text{NOT IN}\} \equiv \{<>\text{ALL}\}}$

★ If inner query is empty, then $\boxed{\text{NOT IN returns true.}}$

Q. A

id	Name	age
12	A	60
15	S	24
99	R	11

B

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

C

ID	Pno	Age
10	2200	02
99	2100	01

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

Ans. $\boxed{7}$ [no of tuples returened]

(2) Select A.ID from A where a.age > ALL (Select B.age from B where B.name='A')

empty

Ans $\boxed{3}$ [no. of tuples returned]

~~(1) 1 99 R 11 2100 01~~

Correlated Nested Queries:-

Exists:- returns true only if Inner Query not empty.
Exists(Inner Query Result)

```
⑥ Select c.sid from c
① from catalog c      ⑤                     ③
② where EXISTS(Select * from parts p
              where p.pid = c.pid and   ] ④
              colour = RED);
```

Catalog

sid	pid
s1	p1
s2	p2
s3	p3

parts

pid	colour
p1	R
p2	G
p3	R

Step 1:- Take 1st tuple of catalog & then execute inner query, in this case c.pid = p1, then ~~p.pid = p~~ one tuples from inner query satisfies, & hence set is not null ∴ exists returns true, ∴ c.sid = p1 is O/P.

Step 2:- Take 2nd tuple of catalog & repeat step 1 this time inner query results in empty set & hence exists returns false. ∴ s2 is not o/p.

Step 3:- Take 3rd tuple & repeat

O/P:-
S1
S3

* Correlated nested query takes more time to execute than independent queries

Q.
```
Select c1.sid
from catalog c1
where NOT EXISTS(Select p.pid
                 from parts p
                 where NOT EXISTS(Select c2.sid
                                  from catalog c2
                                  where c2.pid =
                                        p.pid
                                  and c2.sid =
                                      c1.sid));
```

≥1 (may be all)

(a) sid of suppliers who supply some parts
(b) " " " " " proper subset of parts.
(c) " " " " " all parts.
(d) " " " " do not supply any part.

for this we have to differentiate b/w 4 options, take data accordingly

(C1)

sid	pid
S1	p1
S1	p2
S1	p3
S2	p1

(P)

pid
p1
p2
p3

(C2)

sid	pid
S1	p1
S1	p2
S1	p3
S2	p1

S1

p2, p3

(S1) p2

For above problem
(a) → S1, S2
(b) → S2
(c) → S1
(d) → φ
the answer comes out to be (c)

Comparison with NULL:-

Null : Unknown or ~~unexi~~ doesn't exist
Null is non-zero & no two NULL's are equal.

EID	Ename	PPno
E1	A	NULL
E2	A	P5
E3	B	NULL

⇒ Null is random ASCII characters Assigned DBMS.

Eid's which have no passport

```
Select eid
  from emp
    where ppno is NULL;
```

is/is NOT : Compare with NULL values.

Comparison with Regular Expression :-

'%' ⇒ 0 or more characters
'_' ⇒ exactly any one character

- Names starts with 'D' & ends with 'A' & atleast 5 characters

  D%_%_%_%A

- Name starts with 'R' :- 'R%'

- Name starts with 'A_' & ends with '_B' & atleast 6 characters.
  for this use escape character '/'.
  'A/_%_ _%_/_B'.

Like/Not Like:-

Compare with regular expression:-

```
select sname
    From stud
      where sname LIKE 'D___%A';
```

Foreign Key :-

Student (Referenced Relm)

sid	sname	login
S1	A	@
S2	A	@
S3	B	@
S4	C	@

Enrolled → Referencing Relm.

sid	cid	fee
S1	C1	-
S1	C2	-
S2	C2	-

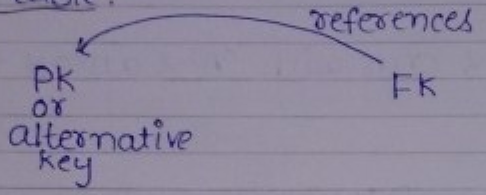
★ Deletion of (S1,A,@) cant be done before deletion of details of S1 in enrolled table.

★ (S4,C,@) can be deleted.

★ (S10,C5,-) is not allowed to insert into enrolled table because S10 is not student table.

★ (S4,C3,-) is allowed to be inserted into enrolled table.

Foreign key is a set of attributes that references primary key or alternative key of the same table or some other table.



Emp	EID	Ename	SubID
	E1		E2
	E2		NULL
	E3		E2
	E4		E3

we can't insert E6 as SubID

SubID is the foreign key referencing EID of the same reln.

Refrential Integrity Constraints

. Referenced Reln.:- [Student]

(i) Insertion :- No voilation

(ii) Deletion :- May cause voilation

(a) ON DELETE NO ACTION :- (if ~~voilation~~ refrential integrity voilation occurs because of deletion from referenced reln, the corresponding deletion is prohibited).

(b) ON DELETE CASCADE :-

~~Foreign~~ If referential integrity voilation occurs, corresponding tuples are deleted from both referenced reln. & referencing reln.

(from above example of emp, if we delete E2, then SubID with E2 are also deleted, so E1 & E3 are also deleted, & then we delete SubID with values E1 & E3, ∴ E4 is delete, so complete table is deleted.)

(c) ON DELETE SET NULL:-

- Deletion takes place only when Foreign key is allowed to have NULL values, otherwise deletion doesn't take place.
- Deletion is allowed from referenced reln. only if corresponding Foreign Key attribute is allowed to have NULL values.
- If Foreign key is a part of primary key or NOT NULL attribute ON DELETE SET NULL = ON DELETE NO ACTION.

ON DELETE SET NULL on emp table :- when we try to delete tuple with eid = E2.

∴ table:-

EID	ename	SU[illegible]
E1		NULL
E3		NULL
E4		E3

(iii) Updation:- (Referenced Attributes value Updation)

Updation means updation of primary key or candidate key which is referenced by F.K.

(May Cause Voilation):-

(a) ON Update NO ACTION → if causes voilation, then dont delete.

(b) ON Update CASCADE → if causes voilation, update the foreign key to new value.

(c) ON Update SET NULL. → if causes voilation, then update F.K. to NULL (if F.K. can be set NULL) & degenerates to ON update No Action when F.K. can't be set NULL.

[2] Referential Integrity Constraints for Referencing reln.

(a) Insertion :- May Cause voilation.

(b) Deletion.:- No voilation

(c) Updation:- (Referencing Attribute Update) May Cause voilation.

If child reln. (referencing reln.) operations causes voilation then corresponding operation is restricted.

Date

02.09.12

A	B
~~2~~	~~4~~
~~3~~	~~5~~
~~5~~	~~2~~
~~9~~	~~2~~
6	7
7	6

A: Primary key
B: FK Referencing to A (ON DELETE CASCADE)
if Tuple (2,4) is deleted

Q. $R(\underline{A}BC) \quad S(\underline{D}E)$

'C' is a foreign key references relation S
which of the following RA produce always empty result.

1. $\pi_D(S) - \pi_C(R)$

[✓ 2.] $\pi_C(R) - \pi_D(S)$

PK	FK
1	1
2	1
3	2
4	3
5	3
	4

PK = { }
FK = { ... }

$\boxed{FK \subseteq PK}$

$\therefore \boxed{\begin{array}{l} PK - FK \neq \phi \\ FK - PK = \phi \end{array}}$

Q.
Book (Title, Price)
· No two books are same price.
What is the o/p of the following SQL query?

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

B

Title	Price
~~X~~	~~10~~
Y	20
Z	30
D	40
F	50
G	60

5<5   4<5

T

Title	Price
X	10
Y	20
Z	30
D	40
F	50
G	60

(a) Titles of 4 most expensive books
[✓ (b)] Titles " 5 " " "
(c) Title " 4th " " "
(d) " " 5th " " "

☆ To get the most expensive book we put '=0' in place of '<5'.

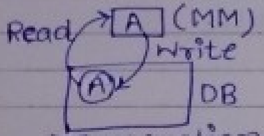
☆ To get 4th most expensive book we put '=3' in place of '<5'.

☆ To get least expensive book we put T.price < B.price.

Transaction & Concurrency Control:-

Transaction:-
Set of logically related operations to perform a unit of work.

- Read (A): Accessing the dataitem from DB to MM(programmed) variab
- Write (A) : Updation of Dataitem into DB.
- Dataitem : DB resource:
  - record
  - block
  - table
  - DB.



Transaction

e.g. R(A)

A=A+10 ← This updation takes place in main memory.

W(A) ← ~~The~~ Updated into the database.

W(B) → Setting of Dataitem Directly into Database irrespective of previous value (Blind Write operation) [i.e. w/o reading the data, we just overwrite the previous data.]

- Commit :- Transaction executed successfully (Transaction committed means Transaction Terminated.)

- Integrity of the trans, Trans should preserve ACID properties

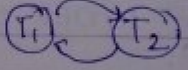
A: Atomicity: Execute all operations or none of them.

e.g. Trans 500Rs from A to B.

Trans
R(A)
A=A-500
W(A)
R(B)
B=B+500
W(B)
Commit

← If transaction failed here, then atomicity is voilated.

Failure Reasons:-
(1) Power Failure
(2) S/W Crash
(3) H/W Crash (DISK CRASH)
(4) Concurrency Control of DBMS/OS may kill Transaction.

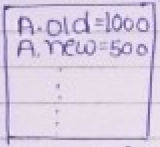


Transaction in deadlock

Recovery Management Components :-

- Rollback the transaction or (Abort) :-
  it is the process of undoing the modification that were done until failure positionpoint.
- Transaction Log :-
  → Activities of transaction

T1.log



Maintained by recovery management component until commit/rollback. [this is stored in secondary memory.] log file

Transaction log is required to perform rollback opn.

Durability :-

Transaction should be able to recover under any case of failure.

- RAID architecture
  (Redundant Array of Independent Disks) :-
- RAID-0 :- No redundant disk (high possibility of failure).
- RAID-1 :- Image Disks (Same DB, log files are maintained in independent Disks), by independent we mean that by failure of one disk, other disk doesn't fail & have no effect due to its failure.

☆ If Transaction failed before Commit, then Atomicity & Durability comes into the picture.

Isolation :-

- Two or more than two transactions are executing concurrently.

T_1(A) : Trans 500Rs from A to B. $r_1(A)\ w_1(A)\ r_1(B)\ w_1(B)$

T_2 : Trans display total of A, B $r_2(A)\ r_2(B)$

Schedule :-

Time order sequence of two or more transaction.

- Serial Schedule :- After 'commit' of one transaction, only then start the other transaction.

- Concurrent Schedule :- Interleaved execution or Simultaneous execution of two or more transaction.

Serial Schedule:-

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

$T_1 \rightarrow T_2$ (serial)

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

$T_2 \rightarrow T_1$ (serial)

$n! \rightarrow$ serial schedules are possible with 'n' transactions.

- Every Serial Schedule is consistent.
- Throughput of system is very less (poor resource utilization).

Concurrent Schedule:-

T_1	T_2
$R_1(A)$	
$W_1(A)$	
	$R_2(A)$ *
	$R_2(B)$
$R_1(B)$	
$W_1(B)$	

[Inconsistent Schedule]

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

[Consistent Schedule]

* This read $R_2(A)$ is different from 2nd serial schedule's $R_2(A)$ because $R_2(A)$ of 2nd S.S. is reading from initial value of DB.

- Throughput increases.
- Inconsistent Schedule.
- Concurrent exec^n of 2 or more than 2 transaction may result in inconsistency.
  To resolve this issue, we use concurrency control componeny.
- Concurrency control component is responsible for avoiding inconsistent concurrency control.

★ For the schedule to be consistent, the concurrent schedule behaviour must

- 1st Concurrent Schedule (Non-Serializable) because it is reading $R_2(A)$ after updation & $R_2(B)$ before updation which is not happening in any Serial Schedule.
- 2nd Concurrent Schedule (equal to $T_1 \rightarrow T_2$ Serial Schedule)

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

Equal to
$T_2 \rightarrow T_1$ Serial Schedule.
(Serializable Schedule).

Serializable Schedule:-

Concurrent execution of 2 or more transaction should be equal to any serial schedule.
(Schedules are equivalent & not equal, because order of execution differs.)

Isolation says that:-
★ Concurrent Schedule should be serializable schedule.

Q
$T_1 : R_1(A)\ W_1(A)\ R_1(B)\ W_1(B)$
$T_2 : R_2(A)\ R_2(B)$
How many concurrent schedules are possible.
Ans. $^6C_4$

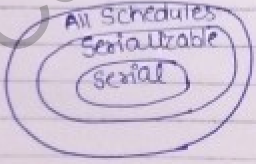
$\frac{6!}{2!\,4!}$

— — — — — —

General Formula:-

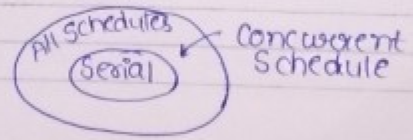
- T_1, T_2 Transaction Consists m, n operations each
  no. of concurrent schedules $= {}^{(m+n)}C_n$

- T_1, T_2, T_3 Transaction consist m, n, p operations each
  no. of concurrent schedules $= {}^{m+n+p}C_m \cdot {}^{n+p}C_n$



★ Serial Schedules are Serializable, but not vice versa.

★ k



- Every Serializable Schedule is not serial but result of serializable schedule is equal to any serial schedule.

Consistency:-

Before & After execution of transaction DB should be consistent. Criteria for consistency:-

- Schedules should be recoverable.
- Schedules should be serializable.

(Both concurrency control & recovery management is used in consistency property.)

T_1	T_2
	$R_2(A)$
$W_1(A)$	
$W_1(B)$	
	$R_2(B)$

Check whether the schedule is serializable or not?

T_1	T_2
$W_1(A)$	
$W_1(B)$	
	$R_2(A)$
	$R_2(B)$

$T_1 \rightarrow T_2$

T_1	T_2
	$R_2(A)$
	$R_2(B)$
$W_1(A)$	
$W_1(B)$	

$T_2 \rightarrow T_1$

- The schedule is not equivalent to $T_1 \rightarrow T_2$.
  (because
- The schedule is not equivalent to $T_2 \rightarrow T_1$.

Q.

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

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

$T_1 \rightarrow T_2$

→ The schedule is equal to $T_1 \rightarrow T_2$, hence the schedule is serializable.

Q.

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

⟶

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

$T_2 \rightarrow T_1$

this is equivalent to $T_2 \rightarrow T_1$.

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

this B must be read initially from DB, & not overwritten value.

★ Every Read should be same & every final updation of data item should be same.

Q.

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

because of $R_2(B)$, the schedule is not equivalent to serial schedule $T_1 \rightarrow T_2$.

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

$T_2 \rightarrow T_1$

In original schedule B is finally written by T_2 but in schedule $T_2 \rightarrow T_1$ B is finally written by T_1, hence not equivalent.

Problems because of concurrent execution:-

(1) RW problem [Write after read problem]

"Transaction T_2 updates data item A which is already read by uncommitted Transaction T_1.

(Simultaneous Read Write operations).

T_1	T_2
R(A)	
	W(A)

T_1	T_2
R(A)	
Commit	
	W(A)

→ not simultaneous read write opn.

example :-

Library DB:

A: no. of copies of DBMS Text book.

```
R(A):
if (A>0)
 {A=A-1;
  W(A)
  Commit
 }
else "no-book"
```

	T_1	T_2	
10 ←	R(A)		Let A=10 initially
10 ←	if (A>0)		
(A is in main memory).		R(A)	→ A is in main memory.
		if (A>0)	
		{A=A-1	
		W(A);	→ A=9 [written in DB]
		Commit	
		}	
9 ←	A=A-1		
A=9 [Written in DB] ←	W(A)		
	Commit		

This is non-serializable schedule.

(This problem occurs because of simultanous read-write Op^n.)

★ A problem is said to be read-write problem only if

(a) simultaneous R/W opn should exist.

(b) schedule is non-serializable.

Schedules having read write op^n may be serializable.

e.g.

T_1	T_2
R(A)	
W(A)	
	R(A)
	W(A)
	commit
Commit	

→ This schedule is serializable so it voilates (b), but follows (a), so no read-write problem.

(Dirty Read)

(2) Write-Read Problem :- {Read after write problem}

Transaction T_2 reads data item A which is updated by uncommitted transaction T_1.

T_1	T_2
W(A)	
	R(A)
Commit	

Uncommitted Read.

T_1	T_2
W(A)	
Commit	
	R(A)

} Not simultaneous WR operations.

Writ

*A problem is Write Read problem :-

(1) Uncommitted read operation should exist.

(2) Non-serializable schedule.

e.g.

T_1	T_2
$R_1(A)$	
$\boxed{W_1(A)}$	
	$\boxed{R_2(A)}$
	$R_2(B)$
$R_1(B)$	
$W_1(B)$	

→ Non-Serializable

→ Uncommitted read

So, write-read problem exist.

T_1	T_2
$R_1(A)$	
$\boxed{W_1(A)}$	
	$\boxed{R_2(A)}$
$R_1(B)$	
$W_1(B)$	
	$R_2(B)$

→ This is Serializable.

Write-Write Problem

T_1	T_2
W(A)	
	W(A)

Transaction T_2 updates data item A which is already updated by uncommitted transaction T_1

⇒ Simultaneous WW opn.

WW problem exists:-

- Simultaneous WW operations.
- Non-Serializable.

T_1	T_2
$W_1(A)$	
	$W_2(A)$
	$W_2(B)$
$W_1(B)$	

→ Non-serializable

→ WW problem

Lost Update Problem:-

(It is possible in non-serializable also.)

It is possible even though the schedule is serializable.

	T_1	T_2
A=10 ←	W(A)	
		W(A) → A=20
	x	

Transaction T_1 fails

initial value of A=~~0~~ ~~10~~ 20

when T_1 fails, rollback manager uses log of T_1 to rollback, so the updates done by T_2 are lost.

Lost update problem is possible if simultaneous Write-Write opⁿ exists.

Lost update (WW problem) problem

★Every

Classification Schedule

Recoverability

- Irrecoverable
- Reoverable
- cascade less record
- Strict Recoverable

doesn't eliminate lost update problem.

Eliminate lost Updates

Serializability

- Conflict S.S.
- View S.S.

★ Schedule is said to be consistent only if schedule is strict recoverable schedule, as well as serializable.

Recoverability Classification

(1) Irrecoverable Schedule:

- Rollbacking of committed transaction is irrecoverable.
- Irrecoverability may be possible only if uncommitted reads exist.

T1	T2
W(A)	
	R(A) ← uncommited read
	Commit
*	

Rollback possible

(T_2 is reading A written by transaction T_1)

↑ can't be rolled back because it is committed.

T1	T2
W(A)	
	R(A)
*	

failed

→ both can be rolled back.

★ If transaction T_2 reads the dataitem A which is updated by T_1 & T_2 committed before commit or rollback of T_1, then schedule is said to be irrecoverable.

T1	T2
W(A)	
	R(A)
	Commit
C/R	

C:- Commit
R:- rollback.

T1	T2
W(A)	
	R(A)
	C2
C1	

⇓
unsafe

because C1 might fail & hence no rollback of T_2.

T1	T2
W(A)	
	R(A)
	R2
C1/R1	

↑Recoverable

T1	T2
W(A)	
	W(A)
	C2
C1	

→ no uncommitted read (Recoverable Schedule)

(⟵? doubt)

– Lost update problem

T1	T2
R(A)	
	W(A)
W(A)	
R(A)	
C1	
	C2

→ no uncommitted read ∴ recoverable.

Recoverable Schedule:-

If transaction T_2 reads data item A which is updated by uncommitted transaction T_1, then commit opn of T_2 should be delayed until comit or roleback of T_1

T_1	T_2
W(A)	
	$R_2(A)$
C1/R1	
	C2

T_1	T_2
W(A)	
	R(A)
	R(B)
W(B)	
C1	
	C2

→ Even ~~if~~ recoverable schedule might suffer:- from non-serializability (WR, RW, WW problems are possible).

- Lost Update also possible.

Cascading Rollback Problem:-

T_1	T_2
W(A)	
	R(A)
	W(A)
R(A)	
C1	
	C2

failure of one transaction results rollback of set of other transaction.

T_1	T_2	T_3	T_4
W(A)			
	R(A)		
	W(A)		
		R(A)	
			R(A)
×			

fails.

→ Recoverable

* If T_1 fails, then ~~T_2~~ we have to rollback transaction depending on T_1 which is T_2 & then we have to rollback transactions depending on T_2 which are T_3 & T_4.

* Wastage of CPU time & I/O access.

Cascadeless Rollbacking Recoverable Schedule:

- Recoverable Schedule

&

- No cascading rollbacks.

T_1	T_2
W(A)	
C/R	
	R(A)

Uncommitted Read opn are not allowed. for Cascadeless Rollbacking.

e.g.

T₁	T₂
R(A)	
	W(A)
	W(B)
W(B)	
W(C)	
C1	
	R(C)
	C2

This schedule is cascadeless rollbacking recoverable.

Cascadeless Rollbacking Recoverable Schedules are:

not free from:-
1. WW problem
2. RW problem
3. Lost update

free from:-
1. WR problem
2. Cascading Rollback

Strict Recoverable Schedule:-

(Cascadeless rollbacking recoverable) & (No lost update problem / No simultaneous WW Opn.)

T₁	T₂
W(A)	
C/R	
	R(A)

T₁	T₂
W(A)	
C/R	
	W(A)

T₁	T₂
W(A)	
C/R	
	W(A)/R(A)

Strict Recoverable Condⁿs.

If transaction T₁ updates data item A, other transaction T₂ is not allowed to read or write data item A until commit or rollback of T₁.

not free from:-
RW problem

free from:-
WW, WR, lost update, cascading rollback problem.

$r_3(y)$

$W_1(x)$

C1

$W_2(z)$

$W_3(y)$

$W_2(y)$

C3

C2

Serializability Classification :-

[1] Serializability Classification

(i) Conflict Serializable Schedule:-

Conflict Pairs:- (i) $R_i(A)$ $R_j(A)$:- Non-conflict pairs

T_1	T_2
$R_i(A)$	
	$R_j(A)$

S1

T_1	T_2
	$R_j(A)$
$R_i(A)$	

S 2

⟶ are similar.
S1 ≡ S2

(ii) $R_i(A)$ $W_j(A)$: Conflict Pair

T_1	T_2
$R_i(A)$	
	$W_j(A)$

S_1

T_1	T_2
~~W~~	$W_j(A)$
$R_i(A)$	

S_2

$S_1 \neq S_2$

(iii) $W_i(A)$ $R_j(A)$: Conflict Pair

(iv) $W_i(A)$ $W_j(A)$: Conflict pair

(v) $R_i(A)/W_i(A)$, $R_j(B)/W_j(B)$: Non-conflict Pai

Pair of opn is said to conflict only if there is atleast one write op^n
& (ii) on same data item.
(iii) on diff. transactions.

Conflict Equal Schedule :-

If conf S_j results after swapping of non-conflict pairs in S_i then S_i & S_j are said to be conflict equal schedules.

T_1	T_2
R(A)	
W(A)	
	R(A)
	R_2(B)
R_1(B)	
W(B)	

S1

T_1	T_2
R(A)	
W(A)	
	R(A)
R_1(B)	
	R_2(B)
W(B)	

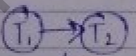
S2

S_1 conflict equal to S2.

Schedule should be
- Conflict equal should ~~be~~ to any serial schedule.

- Any One of the serial schedule should be conflict equal to given schedule (only then we can say given schedule is conflict ~~equa~~ serializable (i.e. after swapping non-conflict pairs, we must get a serial schedule.)

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



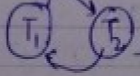
conflict serializable

non-conflicting pairs

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

$T_1 \rightarrow T_2$ (conflict equal to $T_1 \rightarrow T_2$)

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

→  → cycle
(not conflict serializable)

Precedence Graph

G = (V, E)

- Vertices: Transactions of the schedule.

Edges:- conflict pair precedence order.

$T_i \rightarrow T_j$ $W_i(A)$ $W_j(A)$
$W_i(A)$ $R_j(A)$
$R_i(A)$ $W_j(A)$

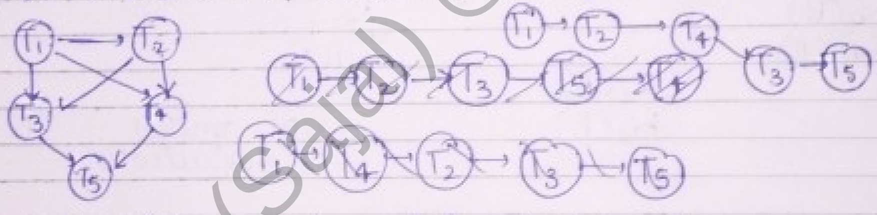
Testing Condition:-

(a) if precedence graph cyclic then not conflict Serializable Schedule.

(b) if precedence graph is acyclic then it is conflict serializable.
Equivalent serial schedule is based on topological order of acyclic precedence graph.

Topological Order:-

(1) Visit Vertex (V) with indegree '0' & delete 'V' from G.
(2) Repeat (1) until G becomes empty.



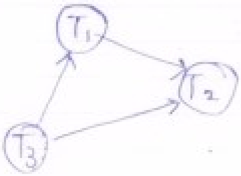
T_1
T_2
T_3
4
T_5
T_1 → T_2 → T_4 → T_3 → T_5
T_1 → T_2 → T_3 → T_5 → T_4
T_1 → T_4 → T_2 → T_3 → T_5

★ No. of conflict equal serial schedules is equal to no. of topological orders of acyclic precedence graph.

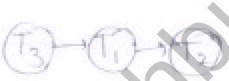
Q. S_1: ~~$r_3(y)$ r_3~~

$r_5(y)$

T_1	T_2	T_3
		$r_3(y)$
		$r_3(z)$
$r_1(x)$		
$w_1(x)$		
		$w_3(y)$
		$w_3(z)$
	$r_2(z)$	
$r_1(y)$		
$w_1(y)$		
	$r_2(y)$	
	$w_2(y)$	
[illegible]	$r_2(x)$	
	$w_2(x)$	

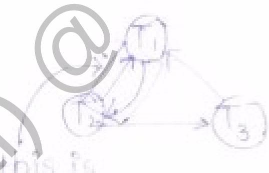


Conflict Serializable

[illegible] 

S2:

T_1	T_2	T_3
	$r_2(A)$	
$r_1(B)$		
	$w_2(A)$	
		$r_3(A)$
$w_1(B)$		
$w_1(A)$		
	$r_2(B)$	
	$w_2(B)$	

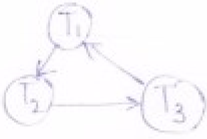


this is also cycle.

not conflict serializable

S3:

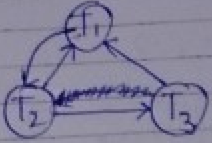
T_1	T_2	T_3
		$w_3(A)$
$r_1(A)$		
$w_1(B)$		
	$r_2(B)$	
	$w_2(C)$	
		$r_3(C)$



cycle

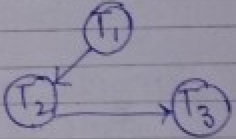
∴ not conflict serializable.

S4:	T_1	T_2	T_3
		$r_2(z)$	
		$r_2(y)$	
		$w_2(y)$	
			$r_3(y)$
			$r_3(z)$
	$r_1(x)$		
	$w_1(x)$		
			$w_3(y)$
			$w_3(z)$
		$r_2(z)$	
	$r_1(y)$		
	$w_1(y)$		
		$w_2(x)$	



Non-conflict Serializable.

S5:	T_1	T_2	T_3	T_4
	$R_1(A)$			
		$R_2(A)$		
			$R_3(A)$	
				$R_4(A)$
	$W_1(B)$			
		$W_2(B)$		
			$W_3(B)$	



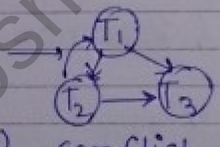
T1 → T2 → T3 → T4

T4 → T1 → T2 → T3

T1 → T4 → T2 → T3

T1 → T2 → T4 → T3

T_1	T_2	T_3
R(A)		
	W(A)	
W(A)		
		W(A)



not conflict serializable.

Equality condn :-

① Every read should be same.

② Last updation must be done by same Transaction

Q.

T_1	T_2	T_3
$R_1(A)$		
	$W_2(A)$	
$W_1(A)$		
		$W_3(A)$

S_1

Not conflict ~~se~~ serializable.

T_1	T_2	T_3
$R(A)$		
$W_1(A)$		
	$W_2(A)$	
		$W_3(A)$

S_2 : T_1, T_2, T_3 → Serial

S1 not conflict equal to S2

S1 equivalent to S2

★ → Serializable schedule (because it is equivalent to a serial schedule) but not conflict serializable schedule.

Q.

T_1	T_2	T_3
$W_1(A)$		
	$W_2(A)$	
		$W_3(A)$

S_1

T_1	T_2	T_3
	$W_2(A)$	
$W_1(A)$		
		$W_3(A)$

S_2

$S_1 = S_2$

but S_1 & S_2 are not conflict ~~equal serial~~ equal schedule.

★ if (acyclic precedence graph)
    conflict serializable & hence serializable
else
    not conflict serializable & (may or may not be serializable).

only sufficient but not necessary

View Serializable Schedule Testing Condn. :-

if (VSS condn)
    View Serializable & hence Serializable
else
    not view serializable & ~~Not~~ non-serializable

Condn → predicate [if L is ~~f~~ true, R is true
if L if false, R may or may not be false]

sufficient but not necessary.

Condn. ⟷ Predicate [Sufficient & necessary.]

View Serializable Schedule

View equivalent schedule should be any serial schedule. (one of the serial schedule should be view equivalent to the given schedule.)

View Equivalent Schedule Condn :-

- S1 & S2 are view equivalent only if :-

(1) If transaction T_i reading dataitem 'A' from initial DB in S1, then S2 should ~~also~~ in S_2, Trans T_i should read dataitem A from initial DB only.

T_1	T_2	T_3
R(A)		
	R(B)	
		W(B)

S_1

T_1	T_2	T_3
R(A)		
		W(B)
	R(B)	

S_2

because in S_2 R(B) reads B after W(B) ∴ $S_1 \neq S_2$

(2) If transaction T_i reading dataitem A which is updated by T_j in S_1, then Transaction T_i should read 'A' which is updated by T_j in S_2 also.

T_1	T_2	T_3
W(A)		
	R(A)	
W(B)		
		W(B)
	R(B)	

S_1

T_1	T_2	T_3
W(A)		
	R(A)	
W(B)		
	R(B)	
		W(B)

S_2

∴ $S_1 \neq S_2$

(3) If transaction T_i have final updates of dataitem 'A' in S_1, then transaction T_i should have final update of dataitem 'A' in S_2 ~~fi~~ also.

T_1	T_2	T_3
W(A)		
	W(A)	
		W(B)
	W(B)	
W(B)		

S_1

A by T_2
B by T_1

T_1	T_2	T_3
W(A)		
	W(A)	
		W(B)
W(B)		
	W(B)	

S_2

A by T_2
B by T_2

$S_1 \neq S_2$

☆ Two Schedules are equal only if ~~&~~ all 3 condn are satisfied.

Date

08.09.12

Serializability :-

The schedule is conflict Serializable

- → Acyclic Precedence Graph [if graph is cyclic, then it is not conflict serializable]
- → If graph is acyclic, we can't say anything.

T_1	T_2	T_3
W(A)		
	R(B)	
	R(A)	
	W(B)	
W(B)		
		W(B)

This can make 6 serial schedules.

1. $T_1 \rightarrow T_2 \rightarrow T_3$
2. $T_1 \rightarrow T_3 \rightarrow T_2$
3. $T_2 \rightarrow T_1 \rightarrow T_3$
4. $T_2 \rightarrow T_3 \rightarrow T_1$
5. $T_3 \rightarrow T_1 \rightarrow T_2$
6. $T_3 \rightarrow T_2 \rightarrow T_1$

Final Write

★ A is finally written by T_1 & not anyone else,
∴ T_1 can be anywhere.

★ B is finally written by T_3, but is also written by T_1 & T_2.
∴ T_3 must be at the end.

WR quence

★ $(T_1\ T_2) \rightarrow T_3$

$T_1 : W_1(A)$ $T_2 : R_2(A)$ T_k : Write (A)

$T_1 \xrightarrow{T_k} T_2$ $T_k : \phi$ {no other transac. updating A}

∴ $\boxed{T_1 \rightarrow T_2}$ $\boxed{T_k \text{ must be } \phi.}$

Initial Read ★

Data Item	Initial Read	Write
A	–	T_1
B	T_2	T_1, T_2, T_3

$\boxed{T_2 \rightarrow T_1}$

$\boxed{T_2 \rightarrow T_3}$

as $T_1 \rightarrow T_2$ & $T_2 \rightarrow T_1$

∴ there is a cycle,

∴ it is not serializable.

& also not conflict serializable.

★★ if $T_i \rightarrow T_j$ & $T_j \rightarrow T_i$

∴ Non-Serializable Schedule.

Q.

T_1	T_2	T_3
		R(B)
W(A)		
	R(B)	
	R(A)	
	W(B)	
W(B)		
		W(B)

Final Write ★ T_3 must be done at the end.

& no constraint as such for A.

Initial Read ★

DI	IR	Write
A	-	T_1
B	T_3, T_2	T_1, T_2, T_3

∴ $T_2 \rightarrow T_1, T_3$

$T_3 \rightarrow T_1, T_2$

⇒ $T_2 \rightarrow T_3$

& $T_3 \rightarrow T_2$

∴ it is non-serializable.

Q

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

⟶ non-serializable.

Final Write - of A by T_1

of B by T_1

∴ $T_2 \rightarrow T_1$.

Initial Read :-

	IR	Write
A	~~T_3~~ -	T_1
B	T_2	T_1

∴ $T_2 \rightarrow T_1$

WR :- $T_1 \xrightarrow{T_k} T_2$

& $T_k = \phi$

∴ $T_1 \rightarrow T_2$

& $T_2 \rightarrow T_2$

∴ it is non-serializable.

Q.

T_1	T_2	T_3
	R(B)	
	R(A)	
W(B)		
W(A)		
	W(A)	
		W(A)

Final write:- of A by T_3 (& also by T_1, T_2)
$\therefore T_3$ must be at the end.
of B by T_1 only
$\therefore$ no constraint.

WR opn:- No WR opn.

Initial Read:-

	IR	Write
A	T_2	T_1, T_2, T_3
B	T_2	T_1

$\therefore \left.\begin{array}{l} T_2 \to T_1 \\ T_2 \to T_3 \end{array}\right\}$ of A
& $T_2 \to T_1$ } of B

$\therefore T_2 \to T_1$ & ~~T_3 [illegible]~~ $(T_1, T_2) \to T_3$
$\therefore \boxed{T_2 \to T_1 \to T_3}$
$\therefore$ Serializable.

Q.

T_1	T_2	T_3	T_4
$r_1(A)$			
	$r_2(A)$		
		$r_3(A)$	
			$r_4(A)$
$W_1(B)$			
	$W_2(B)$		
		$W_3(B)$	
			$W_4(B)$

Identify view equal serial schedules.

(1) Final write :-
of A → none (no constraint)
of B → by T_4 $\therefore (T_1, T_2, T_3) \to T_4$

(2) WR:- no WR opn.

(3)

③ Initial read:-

	IR	Write
A	T_1, T_2, T_3, T_4	-
B	-	T_1, T_2, T_3, T_4

∴ no constraint by IR.

$\therefore \underbrace{(T_1, T_2, T_3)}_{\downarrow} \rightarrow T_4$

3! combinations = 6

∴ 6 ser equivalent to 6 view serial schedules.

Q. S:

T_1	T_2	T_3
$R_1(A)$		
	$R_2(B)$	
	$W_2(B)$	
		$R_3(A)$
		$W_3(A)$
$R_1(B)$		
		$R_3(B)$
	$R_2(A)$	
	$W_2(A)$	

Final write:- of A by T_2
of B by T_2

$\therefore (T_1, T_3) \rightarrow T_2$.

Initial Read:

	IR	Write
A	T_1, T_3	T_2
B	T_2	T_2

$\therefore (T_1, T_3) \rightarrow T_2$

WR:- $W_2(B) \rightarrow R_1(B)$

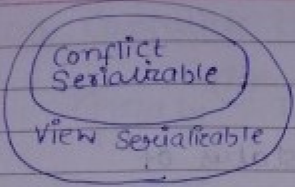
$T_2 \xrightarrow{T_k} T_1$ & $\boxed{T_k = \phi}$

& $W_2(B) \rightarrow R_3(B)$

$\therefore T_2 \xrightarrow{T_k} T_3$ & $\boxed{T_k = \phi}$

as $T_1 \rightarrow T_2$
& $T_2 \rightarrow T_1$

∴ nonserializable.



- Schedule is correct only if :-

(1) Serializable &

(2) Strict recoverable.

T_1	T_2
R(Q) R(P)	R(Q)
R(P) R(Q)	R(P)
if (P == 0)	if (Q == 0)
{ Q = Q+1	{P = P+1
W(Q)}	W(P)}

P = 0, Q = 0 // Initial Values.

Non-Serial interleaved execution

(a) Serializable.

✓(b) Not Conflict S.S. (but not totally correct, as it is also not view serializable).

(c) Not S.S. but View Serializable

(d) Precedence graph can't be drawn.

☆ Precedence graph can be drawn always, ∴ (d) is always false.

☆ $T_1 \rightarrow T_2$:- P = 0
{ Q = ~~0~~1

☆ $T_2 \rightarrow T_1$:- P = ~~0~~1
Q = 0

any non-serial interleaved execution →

```
T1            | T2
R(P)          |
R(Q)          |
              | R(Q)
              | R(P)
              | if(Q=0)
              | {P=P+1
              |  W(P)}
if(P==0)      |
{Q=Q+1,       |
 W(Q);        |
}             |
```

→ P = 1, Q = 1 → which is not equivalent to any serial schedule.

R(Q) → this R(Q) must come after W(Q) which makes it Serial Schedule (but ques. says we have to take non-serial schedule) ∴ (a) is not an option.

Concurrency Control Protocol:-

- Locking
- Timestamp Ordering

Locking Protocols :-

Lock :- Variables used to identify the status of dataitems.

Mutex

CS	⟷	Lock(A)
		dataitem
		Unlock(A)

Transaction

$l_1(A)$ ← grants
$R_1(A)$
$W_1(A)$
$U_1(A)$ ← unlocks
$l_1(B)$ ← denied

Time Out {

$l_1(B)$ ← grants
$R_1(B)$
$W_1(B)$
$U_1(B)$

Shared Exclusive Locking :-

Shared Lock :- [S]

Read only Lock

T_1
S(A) → Shared lock on A
R(A) Transaction T_1 is allowed to read
W(A) → x only, & no write permission.

Exclusive Lock :- [X]

Read/Write Lock

T_1
X(A) → exclusive lock on A
R(A) Transaction T_1 is allowed to
W(A) both read & write.

Lock Compatible table:-

Held by T_i → (A) ← T_j requesting A

	S	X
S	✓	✗
X	✗	✗

← Held by T_i

↑ dataitems requested by T_j.

Shared lock :- S
Exclusive lock :- X

★ To ensure Serializability, non-serializable schedules must not be allowed to execute.

T_1	T_2

X(A)
R(A)
W(A)
U(A)	

	S(A)
	R(A)
	U(A)
	S(B)
	R(B)
	U(B)

X(B)
R(B)
W(B)
U(B)	

→ non-serializable schedule which is being allowed by these locks. (So locks doesn't ensure serializability.)

★ Two Phase Locking:-

Transaction ~~t~~T is allowed to request lock on any dataitem only if no unlock opn is performed by T.

T_1
X(A)
S(B)
S(C)
X(D)

Locking Phase (Growing Phase)

Lock Pt. position of Last lock or first unlock opn

U(C)
X(E) ← not allowed
U(A)
U(B)
U(D)

Shrinking Phase. (Unlocking Phase)

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

☆ If schedule is non-serializable, then not allowed by 2PL.

Q.

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

check if it is non-serial

→ This is Serializable Schedule ($T_1 \rightarrow T_2$).

eg

T_1	T_2
X(A)	
R(A)	
W(A)	
X(B) → U(A)	
	S(A) ← this can only happend if T_1 unlocks A.
	R(A)
R(B)	
W(B)	
U(B)	
	S(B) ← this can only happen if T_1 unlocks B.
	R(B)
	U(B)
	U(A)

This can only happen if T_1 locks B before U(A)

∴ allowed to execute by 2PL.

Q.

T_1	T_2	T_3
	R(A)	
R(A)		
	W(B)	
W(A)		
W(B)		

Q

T_1	T_2	T_3
	R(A)	
R(A)		
	W(C)	
		R(B)
W(A)		
	W(A)	
		W(A)

Check for its serializability :-

Final Write : A by T_3

∴ $(T_1, T_2) \rightarrow T_3$

C by T_2

~~∴ $(T_1, T_3) \rightarrow T_2$~~

~~∴ $T_2 \rightarrow T_3$ & $T_3 \rightarrow T_2$~~

non-serializable.

Initial read :-

	IR	Write
A		

$T_2 \rightarrow T_1 \rightarrow T_3$

∴ Serializable

T_1	T_2	T_3

	X(A)
	R(A)
	S(C)
	U(A)	

X(A)
R(A)
U(A)		

		S(B)
		R(B)

	W(A)	
		W(A)

→ without Lock upgrading (2PL not allowed).

Lock Upgrading technique:-

- Read Op^n should allow only shared lock.
- Shared lock can be upgraded to exclusive lock if transaction has not done any unlock op^n.

T_1	T_2	T_3

	S(A)
	R(A)
	~~X(C)~~	

S(A)
R(A)		

	X(C)
	W(C)	

		S(B)
		R(B)

X(A)
W(A)
U(A)		

	X(A)
	W(A)
	U(A) U(C)	

		X(A)
		W(A)
		U(A)
		U(B)

This X(A) is not allowed by A is shared locked by T_2.

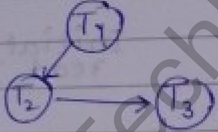
→ not in 2PL.

T_1	T_2
S(A)	
R(A)	
	S(A)
	R(A)
	U(A)
X(A)	
W(A)	
U(A)	

S(A) → This S(A) is allowed because T_1 has share lock on A.

Q.

T_1	T_2	T_3
	R(A)	
R(B)		
	W(A)	
		R(A)
W(B)		
		W(A)
	R(B)	
	W(B)	



∴ conflict serializable.

but not allowed by 2PL.

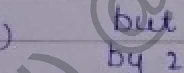
2PL :-

Ensuring Serializability :

★ if Schedule is allowed to execute by 2PL then schedule is conflict serializable but not vice-versa.

eg.

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



not conflict serializable & also not allowed by 2PL.

★ if schedule is not conflict serializable then schedule is not allowed to execute by 2PL.

★ If Schedule is allowed to execute by 2PL, then it is always serializable.

★ Equivalent Serial Schedule is based on order of lock points.

e.g.

T_1	T_2
X(A)	
R(A)	
W(A)	
X(B)	
U(A)	
	S(A)
	R(A)
R(B)	
W(B)	
U(B)	
	S(B)
	R(B)
	U(B)
	U(A)

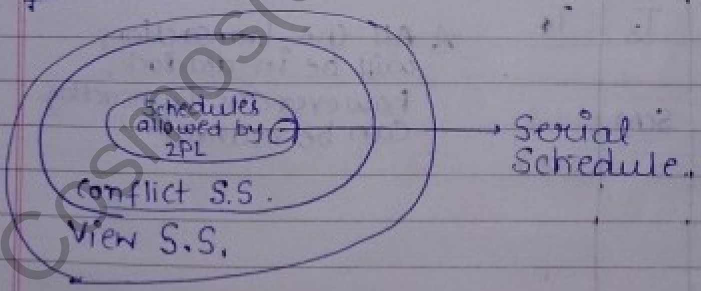
→ last lock points of T_1 & T_2

∴ Equivalent S.S. :- $T_1 \rightarrow T_2$ (acc. to the order of lock points.)

&

T_1	T_2	T_3
	*	
		*
*		

→ Equivalent S.S. :- $T_2 \rightarrow T_3 \rightarrow T_1$



★ Serial ≠ Serializable

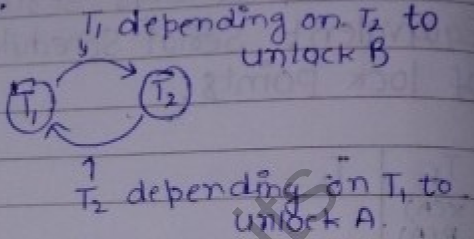
Serial: concurrent execⁿ is not allowed.

Serializable: concurrency is allowed.

Two phase locking protocol:-

1. 2PL restriction
(May cause deadlock).

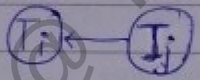
T_1	T_2
S(A)	
R(A)	
	X(B)
	W(B)
denied → S(B)	
R(B)	
	X(A) → denied
	W(A)


T_1 depending on T_2 to unlock B
T_1
T_2
T_2 depending on T_1 to unlock A.

Dependency Graph

Dependency Graph

T_i	T_j
X(A)	
	S(A) ← denied (because T_i already holds the dataitem A)



Transaction T_j required resource held by T_i.

2. 2PL restriction may cause starvation.

T_1	T_2	T_3	T_4
	S(A)		
denied → X(A)			
		S(A)	
	U(A)		
waiting for the next req → X(A)			S(A)
denied (because of T_4) → X(A)			

★ All the transactions will be in deadlock, however one transaction can be starved.

3. May cause irrecoverability.

T1	T2
X(A)	
W(A)	
X(B) U(A)	S(A)
	R(A)
W(B)	
U(B)	S(B)
	R(B)
	U(B), U(A)
	commit
commit	

- Allowed to executed by 2PL.
- Serializable.
- But not recoverable schedule.

Strict 2PL Protocol:-

- 2PL + Strict Recoverability Condn.

T1	T2
W(A)	
C/R	
	R(A)/W(A)

T1	T2
X(A)	
:	
Commit	
U(A)	
	S(A)/X(A)

⇒ Strict Recoverable
⇓
Hold exclusive locks until C/R.

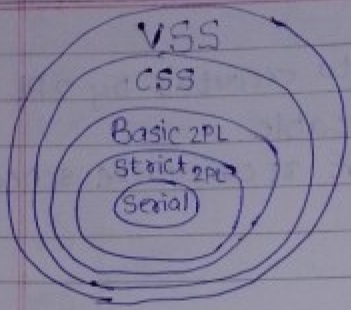
It states that:- Basic 2PL & every exclusive lock held until commit/rollback.

e.g

T
→ X(A)
S(B)
S(C)
X(D)
U(B)
U(C)
Commit
U(A)
U(D)

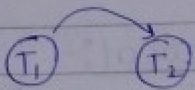
- Strict 2PL:-

→ Ensures serializability (equivalent serial schedules based on lock points.)
→ Ensures Strict recoverability.
→ Deadlocks, starvation still possible in strict 2PL.



Q.

T_1	T_2
R(A)	
	W(A)
R(B)	
	W(B)
	C2
C1	



CSS.

& also serializable ($T_1 \rightarrow T_2$).

Basic 2PL :-

T_1	T_2
S(A)	
R(A)	
S(B)	
U(A)	
	~~S(A)~~ X(A)
	W(A)
R(B)	
U(B)	
	X(B)
	W(B)
	Commit
	U(A),U(B)
Commit	

∴ in basic as well as in Strict 2PL (because of this as exclusive locks can be unlocked after commit)

Q

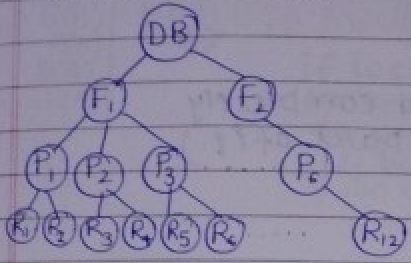
T_1	T_2
X(A)	
W(A)	
~~B~~ X(B) U(A)	S(A)
	R(A)
W(B)	
U(B)	
	S(B)
	R(B)
	U(B)
C1	U(A)
	C2

this can't be written after C1, because then S(A) will be denied.

in 2PL; but not in Strict 2PL.

Multilevel Granularity Protocol:-

(Tree Protocol).



Granularity position of lock (data item size).

High Level Granularity {locking at file level}

e.g. T_1: update $R_1, R_2, \ldots, R_6$:- Lock(F_1)

T_2: ubdate R_2 & R_{12}. :- Lock(F_1) & Lock(F_2).

- Advantage:-

Lock maintainence table consists of no of locks. (easy to maintain)

→ Dis-advantage:- ~~{locking at Record level}~~ Less concurrency level

T_x i:2t,

Low level Granularity :-

T_1: $L_1(R_1)$ $L_1(R_2)$ ..., $L_1(R_6)$

T_2: $L_2(R_2)$ $L_2(R_{12})$

Advantage:- More concurrency level.

Dis-advantage:- Complex to manage lock compatible table (more no. of locks.).

Multilevel Granularity :-

Locking can be allowed at any level.

T_1: $L_1(F_1)$

T_2: $L_2(R_2)$ & ~~$L_2(R_{12})$~~ $L_2(R12)$

e.g.

T_1	T_2
$X_1(F_1)$	
	$x_2(R_2)$ ← denied

as ~~R_2~~ F_1 is locked completely (& R_2 is a part of it.)

T_1	T_2
	$X_2(R_2)$
$X_1(F_1)$	

denied

using BFS for checking whether it can be granted or not (i.e. checking whether any of its descendent is locked or not) will take exponential time [to lock F_1, we need to lock R_2 as well, but it is already locked before, ∴ $X_1(F_1)$ is denied.]

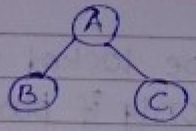
Intention Lock:-

A node N locked by transaction T in intention mode means that any descendents of N can request direct lock by transaction T.

Intension Shared Lock:- [IS]

A node N locked by transaction T_2 is in 'IS' mode means that any descendents of N can request for 'Shared lock' by transaction T.

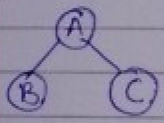
T_1	T_2
IS(A)	S(A)
S(B)	R(B)
R(B)	R(C)
S(A)	
R(A)	



If we applied IS on a node, then to read any descendent, we need to write Shared lock explicitily (like in T_1), but we can directly read descendants if we apply shared lock on A (like in T_2).

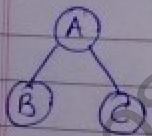
Intension Exclusive Mode (IX):-

A node N locked by transaction T in IX mode means that any descendant of N can request for shared/exclusive mode by transaction T.



T_1	T_2
IX(A)	X(A)
→S(B)	R(B)
R(B)	R(C)
→X(C)	
W(C)	

Shared - Intension Exclusive [SIX]:-



SIX[A]

R(B) ← can directly read data items w/o shared lock because it is available in 'S' in 'SIX'.

R(C) ←

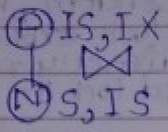
X(C)

W(C) — for writing, we still need to ~~as~~ request for exclusive lock.

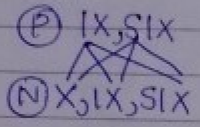
Multilevel Granularity Protocol (Conditions):-

(1) A node N can be locked by transaction T only if parent of N is already locked.

(2) A node N can be locked by Transaction T in S,IS mode only if parent of N is already locked by IX or IS.

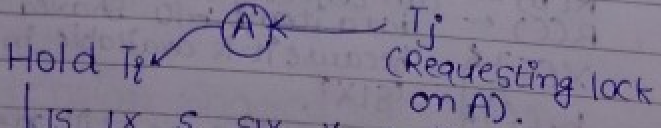


(3) A node N can be locked by Transaction T in X, IX, SIX mode only if parent is already locked by IX or SIX mode by Transaction T.



(4) A node N can ~~be~~ request for lock by Transaction T only if none of the nodes are um

(5) A node N can be locked by 2 diff. transactions only if both locks are compatible



← Hold Ti

↓ requesting Tj

	IS	IX	S	SIX	X
IS	Yes	Yes	Yes	Yes	No
IX	Yes	Yes	No	No	No
S	Yes	No	Yes	No	No
SIX	Yes	No	No	No	No
X	No	No	No	No	No

SIX → IS

S → IS (compatible)

IX – IS (")

S – IX

SIX – IX

1.

T_1	T_2
$IS_1(A)$	
	$IS_2(A)$
S(B)	
	S(B)

← these dataitems can't be updated & ∴ can be in shared mode simultaneously.

2.

T_1	T_2
IX(A)	
	$IS_2(A)$

X(B)
W(B)	

	S(B) ← denied as S(B) not combatible with X(B).
	R(B)

[but if T_1 works on dataitems completely diff. from those on which T_2 is working, then there is no problem.]

Intension lock compatible with Intension lock :-

IS-IX
IX-IS
IS-IS
IX-IX

S-IS
IS-S
S-S

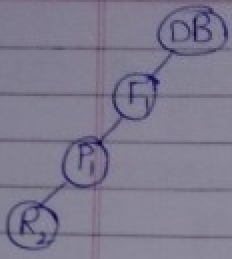
↖ compatible. ↗

3. S – IX

T_1	T_2
S(A)	
	$IX_2(A)$
R(A)	

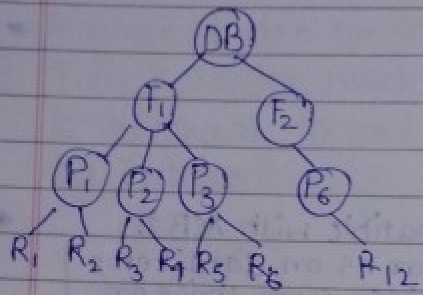
	X(A)
	W(A)

(6) A node N can be unlocked by Transaction T only is none of the descendant is locked by Transaction T.



T
$IX_1(DB)$
$IX_1(F_1)$
$IX_1(P_1)$
$X_1(R_2)$
$U_1(R_2)$
$U_1(P_1)$
$U_1(F_1)$
$U_1(DB)$

Q.



T_1: Update $R_1, \ldots, R_6$
$IX_1(DB), X_1(F_1) \ldots U_1(F_1)\ U_1(DB)$

T_2: Update R2, R12
$IX_2(DB)\ IX_2(F_1)\ IX_2(P_1)\ X_1(R_2)$
$IX_2(F_2)\ IX_2(P_6)\ X_2(R_{12})$
$U(R_{12})\ U(F_2)$
$U(R_2)\ U(P_1)\ U(F_1)$
$U(DB)$

T_1	T_2
$IX_1(DB)$	
$X_1(F_1)$	
	$IX_2(DB)$ → allowed
	$IX_2(F_1)$ → denied

Strict Multilevel Granularity Protocol

Same ①-⑥ steps.

⑦ Hold exclusive locks until commit.

★ Deadlocks & Starvation are still possible.

Time-stamp Ordering Protocol :-

Time Stamp :- Unique value assigned by DBMS to every transaction in ascending order.

e.g. $\overset{10}{\boxed{T_1}}$ $\overset{20}{T_2}$ $\overset{30}{T_4}$ $\overset{40}{T_5}$ $\overset{50}{\boxed{T_3}}$ ...

↑ older ↑ younger

Read Time Stamp (Q) :-

Highest transaction Time Stamp value that has performed R(Q) operation successfully.

A : dataitem

⑩ T_1	⑳ T_2	㉚ T_3	㊵ T_4
R(A)			
W(A)			
		R(A)	
	R(A)		
			W(A)

RTS(A) = $\cancel{0}$ $\cancel{10}$ 30

"Youngest transaction that has performed R(A) opⁿ" [out of T_1, T_2 & T_3 :- T_3 is youngest]

[Initialy RTS(A)=0, means no trans. has read the dataitem A.]

Write Time Stamp (A) :-

Highest transaction Time Stamp value that has performed W(A) operation successfully.

WTS(A) = $\cancel{0}$ $\cancel{10}$ 40

Basic Time stamp Ordering Protocol:-

10	20	30
T_1	T_2	T_3

Concurrent execution should be equal to serial schedule based on TS ordering.

this means that acc. to TS ordering $(T_1 \to T_2 \to T_3)$ so ∴ our concurrent schedule should be equivalent to $T_1 \to T_2 \to T_3$.

10	20	30
T_1	T_2	T_3
	R(B)	
W(A)		
		R(A)
	W(B)	
		W(B)
R(B)		

→ This should be equivalent to $T_1 \to T_2 \to T_3$.

this is not allowed as it is read of B after T_3 writes it which is not happening in case of schedule $T_1 \to T_2 \to T_3$ & hence not equivalent to it, & T_1 is also rollbacked.

Q.

10	20
T_1	T_2
	$R_2(A)$
$R_1(A)$	

this is equivalent to $T_1 \to T_2$.
(i)

10	20
T_1	T_2
	$W_2(A)$
$R_1(A)$	

this is not similar to R(A) of $T_1 \to T_2$ & hence denied & T_1 is rollback.
(ii)
$WTS(A) > TS(T_1)$

10	20
T_1	T_2
	$R_2(A)$
$W_1(A)$	

this is not similar to $W_1(A)$ of $T_1 \to T_2$ & hence denied & T_1 is rollback.
(iii)

10	20
T_1	T_2
	$W_2(A)$
$W_1(A)$	

this is not similar to $T_1 \rightarrow T_2$ & hence denied & hence rollback.

(iv)

① <u>Transaction T_1 issues R(A) opn.:-</u>

(a) if $WTS(A) > TS(T_1)$ then rollback T_1.

(b) otherwise $(WTS(A) \leq TS(T_1))$

allowed to execute $R_1(A)$ opn. by transaction T_1 & set $\boxed{RTS(A) = max(RTS(A), TS(T_1))}$

e.g.

10	20	30
T_1	T_2	T_3
R(A)		
		R(A)
	R(A)	

$WTS(A) = 0$
$TS(T_1) = 10$
$0 > 10$ (false)
∴ R(A) is allowed.
set $RTS(A) = max(0, 10) = 10$.

$WTS(A) = 0$
$TS(T_3) = 30$
$0 > 30$ (false)
∴ R(A) is allowed
set $RTS(A) = max.(10, 30) = 30$

$WTS(A) = 0$
$TS(T_2) = 20$
$0 > 20$ (false)
R(A) is allowed
set $RTS(A) = max.(30, 20) = 30$.

② <u>Transaction T_1 issues W(A) operation:-</u>

(a) if $RTS(A) > TS(T_1)$ then rollback T_1.

(b) if $WTS(A) > TS(T_1)$ then rollback T_1.

(c) Otherwise allowed to execute W(A) op^n by Transaction T_1 &

set $WTS(A) = TS(T_1)$.

Q.

T_1	T_2	T_3
$r_1(A)$		
	$r_2(B)$	
$w_1(C)$		
		$r_3(B)$
		$r_3(C)$
	$w_2(B)$	
		$W_3(A)$

which of the following TS orders are allowed to execute S using BTS ordering protocol.

Rollback

(a) $(T_1\ T_2\ T_3) = (30, 20, 10)$

(b) $= (30, 10, 20)$

(c) $= (20, 10, 30)$

(d) $= (20, 30, 10)$

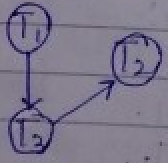
(e) $= (10, 30, 20)$

(no rollback) Check for (a)

~~w TS~~

★ if 'S' is conflict S.S. based on Time Stamp Ordering then 'S' is allowed to execute by TS ordering (Topological order equal to time-stamp ordering.

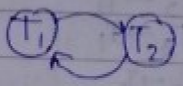
Ans:- make precedence graph for problem:-



the topological sequence is:- T_1, T_3, T_2

- ☆ if 'S' is not CSS, then 'S' is not allowed by BTSO protocol.

20 T_1	T_2 10
R(A)	
	W(A)
W(A)	



- ☆ if 'S' is conflict S.S. & equivalent serial schedule (topological order) is not same as TS ordering, then 'S' is not allowed to execute by BTS ordering protocol.

Option:- (A) $(T_1\ T_2\ T_3) = (30, 20, 10)$

equal to $T_3 \to T_2 \to T_1$

precedences allowed :- $T_3 \to T_2$
$T_2 \to T_1$
$T_3 \to T_1$

T_1	T_2	T_3
* $R_1(A)$		
	* $R_2(B)$	
* W(C)		
		* $R_3(B)$
		$r_3(C)$
	* $W_2(B)$	

* → allowed

(i) because of these $T_1 \to T_3$ (which are not allowed)
∴ T_3 is rollback.

(ii) Now, T_3 is rollback,
carry on with T_1 & T_2 & check if T_1 or T_2 or both are rollback,
now no opn of T_3 will take part in determination of conflict pairs,
i.e. $W_2(B)$ will not be checked with $R_3(B)$.

(b)

T_1	T_2	T_3
$*r_1(A)$		
	$*r_2(B)$	
$*w_1(C)$		
		$*r_3(B)$
		$r_3(C)$
	$*w_2(B)$	

$*$
↓
allowed.

$T_2 \to T_3 \to T_1$

allowed precedence:-

$T_2 \to T_3$

$T_2 \to T_1$

$T_3 \to T_1$

$r_3(B) \to$ gives $T_2 \to T_3$ (allowed)

$r_3(C) \to$ gives $T_1 \to T_3$ (not allowed)

& hence T_3 is rollback

(c)

T_1	T_2	T_3
$*r_1(A)$		
	$*r_2(B)$	
$*w_1(C)$		
		$*r_3(B)$
		$*r_3(C)$
	$w_2(B)$	

$T_2 \to T_1 \to T_3$

∴ allowed precedence

$T_2 \to T_1$

$T_2 \to T_3$

$T_1 \to T_3$

$r_3(B) \to$ gives $T_2 \to T_3$ (allowed)

$r_3(C) \to$ gives $T_1 \to T_3$ (allowed)

$w_2(B) \to$ gives $T_3 \to T_2$ (not allowed)

& hence T_2 is rollback.

(d)

T_1	T_2	T_3
$*r_1(A)$		
	$*r_2(B)$	
$*w_1(C)$		
		$r_3(B)$

$T_3 \to T_1 \to T_2$

∴ allowed precedence

$T_3 \to T_1$

$T_3 \to T_2$

$T_1 \to T_2$

$r_3(B) \to$ gives $T_2 \to T_3$ (not allowed)

& hence T_3 is roll back.

Date

09.09.12

Thomas Write Timestamp Ordering :-

10	20
T_1	T_2
	$W_2(A)$
$R_1(A)$	

Rollback (T_1)

10	20
T_1	T_2
	$R_2(A)$
$W(A)$	

Rollback (T_1)

10	20
T_1	T_2
	$W_2(A)$
~~$(W)(A)$~~	

ignore $W_1(A)$ & continue the execution of T_1.
"Younger Transaction should update dataitem finally".

(1) Read Issue (T):-

$WTS(A) > TS(T)$

Rollback T.

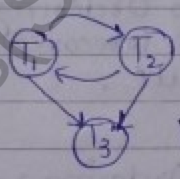
(2) Write Issues (T):-

$RTS(A) > TS(T)$ Rollback (T)

$WTS(A) > TS(T)$ Ignore $W(A)$ operation by Transaction & continue the execution.

Q

10	20	30
T_1	T_2	T_3
R(A)		
	W(A)	
W(A)		
		W(A)



Not CSS but View Serializable Schedule.

Using BTS ordering :- A = 10
$RTS(A) > TS(T)$

Rollback T_1 because of $W_1(A)$.

Using TWR TS Ordering :- No rollbacks, allowed to execute schedule
Equal Serial Schedule: $T_1 \rightarrow T_2 \rightarrow T_3$

★ If schedule is View Serializable schedule & view equivalent serial schedule is based on Time Ordering , then TWR Timestamp Ordering protocol allow to execute the schedule.

BTS Ordering + TWR TS Ordering :-

10 T₁	20 T₂
	W(A)
R(A)	

↑ Rollback (T₁)

30 T₁	40 T₂
	W(A)
R(A)	

↑ Rollback (T₁)

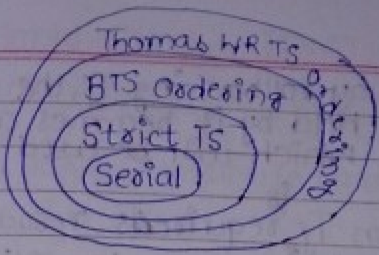
- Deadlock free protocols
- Starvation Possible
- Not free from irrecoveriability.

10 T₁	20 T₂
W(A)	
	R(A)
W(B)	
	R(B)
	C2
C1	

Allowed to execute by BTS Ordering & TWR TS Ordering but is irrecoverable schedule .

Strict Timestamp Ordering :-
Concurrent Schedule should be equivalent to Serial based on TS Ordering & If transaction T_i updates dataitem 'A', other transaction T_j not allowed to R(A)/W(A) until C/R of T_i

- → Strict recoverable
- → Deadlock free
- → Starvation still possible.

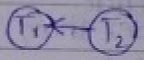


Deadlock Prevention Algorithm :-

- Preventing deadlocks in locking protocols using the timestamp Ordering.

T_1	T_2
S(A)	
	X(A)

Dependence Graph:-

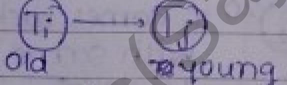


T_2 required resource held by T_1.

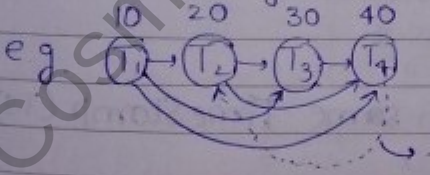
Wait-Die protocol :-

(1) T_i & T_j are any transaction in schedule such that $TS(T_i) < TS(T_j)$

& (2)(i) If Transaction T_i required resource held by T_j, then T_i is allowed to wait.



(ii) If transaction T_j required resource held by T_i, then rollback T_j.



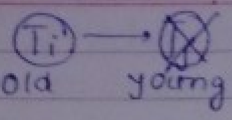
→ possible waiting

→ eg creates cycle which are not allowed, & hence deadlock is avoided.

Wound Wait Protocol :-

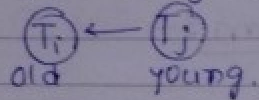
(1) $TS(T_i) < TS(T_j)$

& (2)(i) If Transaction T_i required resource held by T_j, then rollback T_j.

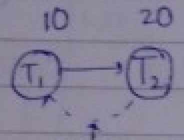


by rollbacking T_j, the resources held by it becomes available which will be used by T_i.

(ii) If transaction T_j is requires resource held by T_i, then T_j is allowed to wait.

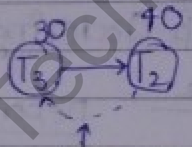


★



because of this, T_2 is rollback.

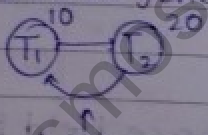
T_2 comes back with TS value greater than previous



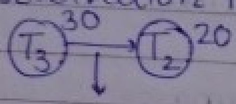
because of this, T_2 is again rollback.

& hence starvation still possible,

but if we are able to start T_2 with same TS value, then starvation is not possible.



because of this, T_2 is rollback



because of this T_3 is rollback.

T_2 comes back with same TS value

★ (iii) So restart T_j with same Time Stamp (TS) value.

Tuple Relational Calculus:- [TRC]

- Non-procedural Query Language :-

① first order logic (Predicate Calculus)

$\in, \vee, \wedge, \neg, \rightarrow, \exists, \forall$, etc.

TRC:-

Atomic formula:-

T ∈ Relation {Tuple variable T should belongs to relation}

T means row of the relation.

eg $\{\forall_x (x \in \text{Stud} \wedge X\,\text{Marks} > 20)\}$

T.A op const → comparing with a constant.

T.A op S.B

attribute A of a tuple.

Tuple Variables:-

Free Tuble Variable	Bounded Tuple variable
Tuple Variable not preceded by Quantifier. $S_1 \in$ Student	preceded by quantifier $\exists$ "There exist" $\forall$ "For All" Tuble Variable preceded by Quantifier bounded Tuple variable. $\exists S_2 \in$ Student $\forall S_3 \in$ Student

★ $\exists R (P(R))$ R: Tuble variable

P(R): formula over tuble variable.

eg $\exists S \in$ Student (S.Marks > 80)

Returns true only if atleast one student scored greater than 80 marks

(If there is no tuple in the relation, then $\exists S$ returns false.)

& $\forall S \in$ Student (S.Marks>80)
returns true only if all ~~[illegible]~~ student scores greater than 80 marks.
returns false if there is atleast one student who scored ≤ 80.

(if tuple set is empty, then ∀ returns true).

Format of TRC:-

{T/P(t)} T: tuple variable
P(t): formula over tuple variable T.

- results Tuples T ~~such that~~ that satisfies P(t) condition.

T ← Select A1,A2
P(t) ← From R
where P

O/P tuple variable (Tuple Variable used before "/") should be free tuple variables.

Q. suppliers(sid, sname, rating)
parts(pid, pname, colour)
catalog(sid, pid, cost)

→ Retrieve suppliers whose rating >10.
$\sigma_{rating>10}$(suppliers)
{S/ S∈ suppliers (S.rating>10)}

$\{T \mid \exists S \in Suppliers(S.rating > 10 \wedge T.sid = S.sid \wedge T.sname = S.sname)\}$

(using

Q→ Retrieve sid of the suppliers who supply some red part.

S/S

$$\pi_{sid}\left(\sigma_{\substack{colour=red \\ \wedge p.pid = \\ c.pid}}(catalog \times part)\right)$$

$$\{T \mid \exists C \in catalog\ \exists P \in parts(C.pid = p.pid \wedge p.colour = red \wedge \underbrace{T.sid = C.sid}_{\text{taken because of this (because of } \pi sid)})\}$$

Or

$$\{T \mid \exists C \in catalog(\exists P \in Parts(P.colour = red \wedge \underbrace{p.pid = C.pid)}_{\text{parts scope ends here}} \wedge \underbrace{T.sid = C.sid}_{\text{for projection}})\}$$

Q retrieve sid of suppliers who supply some red or some green part.

$\pi_{sid}(\sigma_{colour=red \vee colour=green}(parts) \bowtie catalog)$

$\{T \mid \exists C \in catalog\ \exists P \in parts\ (C.pid = p.pid \wedge (p.colour = RED \vee p.colour = Green) \wedge T.sid = C.sid)\}$

→ Sid of the suppliers who supply some red & some green part.

$\{ \quad \mid \exists$

(parts)

$$\{T \mid \exists C1 \in Catalog\ \exists P1 \in Parts\ (C1.pid = p1.pid \wedge p1.colour = RED \wedge \exists C2 \in Catalog\ \exists P2 \in parts\ (C2.pid = p2.pid \wedge p2.colour = Green \wedge C1.sid = C2.sid) \wedge T.sid = C1.sid)\}$$

$$\sigma_{C1.sid = C2.sid}\left(\sigma_{C1pid = p1pid \wedge C1.colour = RED}(catalog \bowtie parts) \times \sigma_{C2.pid = p2.pid \wedge C2\ colour = Green}(catalog \bowtie parts)\right)$$

Or

$$\{T \mid \exists C1 \in Catalog\ \exists P1 \in Parts\ \exists C2 \in Catalog\ \exists P2 \in Parts\ ((C1.pid = p1.pid \wedge p1.colour = RED) \wedge (C2.pid = p2.pid \wedge P2.colour = Green) \wedge (C1.sid = C2.sid) \wedge (T.sid = C1.sid)\}$$

→ Retrieve sid of suppliers who supply atleast two pards.

$$\pi_{c_1.sid}\left(\sigma_{\substack{c1.sid = \\ c2.sid \wedge \\ c1.pid \neq \\ c2.pid}}(catalog \times catalog)\right)$$

$$\{T \mid \exists C1 \in Catalog\ \exists C2 \in Catalog\ (C1.sid = C2.sid \wedge C1.pid \neq C2.pid) \wedge T.sid = C1.sid)\}$$

→ Retrieve sid of the supplier who supplied every part

$$\pi_{sid,pid}(catalog) / \pi_{pid}(parts) =$$
$$\pi_{sid}(catalog) - \pi_{sid}(\pi_{sid}(catalog) \times parts - catalog)$$
$$\Downarrow$$

```
Select C1.sid From catalog C1 where NOT EXISTS
  (Select p.pid from Parts P where NOT EXISTS
   (Select C2.sid from catalog C2 where
       C2.pid = p.pid and C2.sid = C1.sid))
```

Or

```
Select C1.sid from Catalog where NOT Exists(
   Select Pid from parts ]→ gives all part id's
        Except
   Select C2.pid from catalog C2 ]→ gives parts which
   where C2.sid = C1.sid)            are supplied by
                                     C1.sid supplier
```

→ Except gives the difference
(if C1.sid supplier supplied all parts, except returns empty set & NOT-exists for that supplier returns true).

$$\{T \mid \exists C1 \in Catalog\ \forall P \in Parts\ (\exists C2 \in Catalog\ (C1.sid = C2.sid \wedge p.pid = C2.pid) \wedge T.sid = C1.sid)\}$$

$\{S \mid \neg\, S \in \text{Suppliers}\} \equiv \{S \mid S \notin \text{suppliers}\}$

results infinite tables
Unsafe query (it results in infinite set of tuples.)

$$\left\{\begin{matrix}\text{Safe TRC Queries}\\ \text{expressive power}\end{matrix}\right\} \equiv \left\{\begin{matrix}\text{Basic RA}\\ \text{Expressive power}\end{matrix}\right\}$$

$$\left.\begin{matrix}\pi\\ \sigma\\ \times\\ \cup\\ -\\ \rho\end{matrix}\right\} \text{Query using Basic RA also possible to represent safe TRC Query}$$

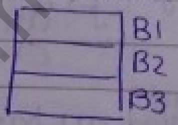
$$\left.\begin{matrix}\bowtie\\ \cap\\ /\\ \bowtie_c\end{matrix}\right\} \rightarrow \text{derived}$$

but
- → Aggregation
- → grouping
- → Outer join (⟕, ⟗, ⟖)

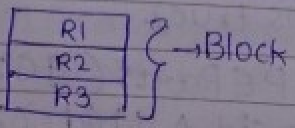
∴ Queries not possible in Basic RA, not even possible in Safe TRC.

Indexing & Physical DB Design:-

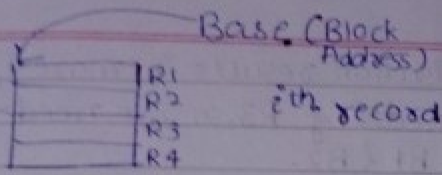
DB file divided into blocks.



Block divided into records:-

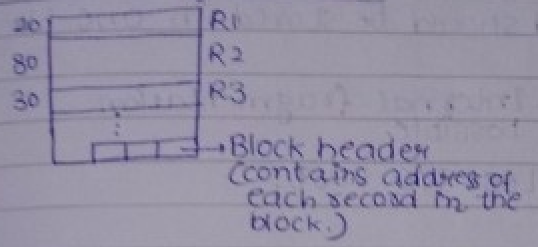


Fixed Length Record



i^{th} record of block = $\text{Base} + \underbrace{(i-1) * \text{Record Size}}_{\text{offset}}$

Variable Length Record



→ Block header (contains address of each record in the block.)

★ Header may be required in fixed length records, e.g. address of next block is saved in the block header

e.g. Block size :- 1250 bytes
block header size :- 250 bytes
Record size :- 200 bytes

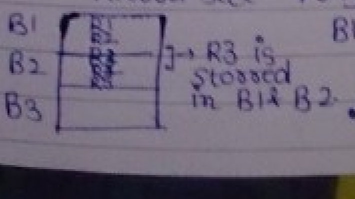
$$\text{Block factor} = \text{no. of records / block} = \frac{\text{block size} - \text{block header size}}{\text{record size}}$$

★ Data transfer rate from secondary memory to main memory is block by block.

Records can be stored in block:

(1) Spanned Organisation
records can be allowed to be stored in two blocks
e.g. Block size = 100 B
Record size = 40 B

(2) Unspanned Organisation



→ R3 is stored in B1 & B2.

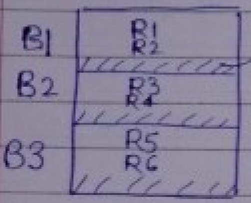
$$\text{Block factor} = \frac{\text{Block Size}}{\text{Record Size}} = 2.5$$

• No internal fragmentation

Spanned Organisation results in more I/O Cost because for accessing R3, so we need to transfer both B1 & B2.

Unspanned Organisation

Complete record should be stored in one block.



→ Internal fragmentation possible.

• Less I/O cost.

★ For fixed length records → unspanned orgn
★ for variable " " → spanned orgn

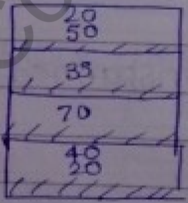
Q. Assume Unspanned blocking & 100 B· blocks, file consist records of 20, 50, 35, 70, 40, 20 bytes, what % of space will be wasted?

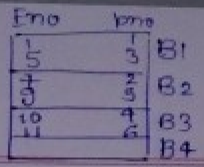
Ans.

20
+50
70   30

35   65
70   30
40
+20   40
165B

$\frac{165}{400} \times 100 = \boxed{41.25\%}$





select *
from emp
where Eno = x;
search key

Emp DB file

Search key:- field used to access data from DB file.

Ordered File:-
Records physically ordered based on search key

Unordered File:-
not physically ordered based on search key.

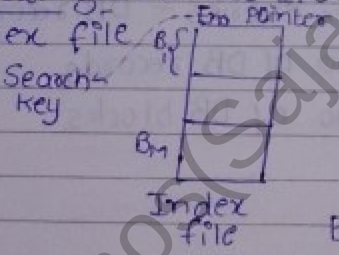
I/O cost :- # of blocks required to transfer from SM to MM to acess some data.

★ Worst case I/O cost (for ordered file) = $\lceil \log_2 N \rceil$ blocks.
N:- no. of blocks of DB file.

Worst case I/O cost (for unordered file) = N blocks.

Indexing (reduce I/O cost)

Index file



Size of DB block
= Size of Index block

Entry In Index File
< search key, pointer>

Entry size of Index file
= size of search key + size of pointer

size of index entry << size of DB record.

★ Block factor of Index block >> Block factor of DB file.

★ no. of index blocks (M) << no. of DB blocks (N)

I/O cost with index = $\{\underbrace{\lceil \log_2 M \rceil}_{\text{to find out which block will contain our record}} + \underbrace{1}_{\text{to access that block}}\}$ Blocks

Categories of Index :

(1) Dense Index :-

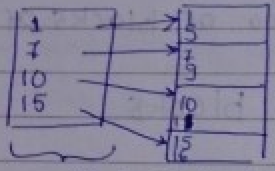
For every DB records, there should be corresponding entry in index file.

1:1 mapping b/w index entries & DB records.

no. of entries in Index files = no. of DB records.

(2) Sparse Index :-

For set of DB records, there exist one entry in the index file.



Sparse Index

1:M mapping b/w index entries & DB records.

- no. of index entries < no. of DB records
- no. of Index entries = no. of DB blocks

Q. Block size 1000 B
& record size = 100 B
Key size = 12 B
pointer = 8 B
no. of DB records = 10,000

(1) How many no. of Dense index blocks req.

Ans. 10000 × (12+8) = 200,000 B

no. of blocks = $\frac{20,000}{1000} = \boxed{20}$ blocks

(2) How many no. of Sparse index blocks req.

Ans. No. of blocks req. for 10,000 DB records = ~~1000×~~ $\frac{10^6}{10^3}$ = 1000 ~~entries~~ blocks

~~no. of~~

1000 entries

1000 × 20B = 20000 B

no. of blocks = $\frac{20,000}{1000}$ = $\boxed{20}$ blocks.

I/O cost (without Index) → $\lceil \log_2 N \rceil$ ordered

→ N unordered

I/O cost (with index) → $\lceil \log_2 M \rceil + 1$ ~~ordered~~

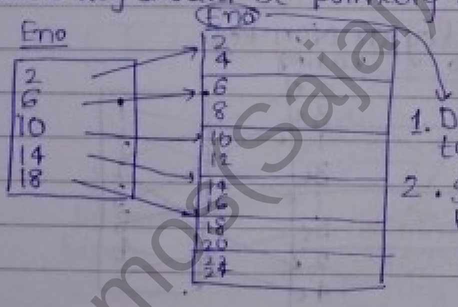
Types Of Index

1. Primary Index :- (default index)

Conditions:-

(1) Ordered file (and)

(2) Search key should be primary key or alternative key.



Primary Index can be dense or sparse.

1. DB is ordered acc to the search key.

2. Search key should be primary or alternative key.

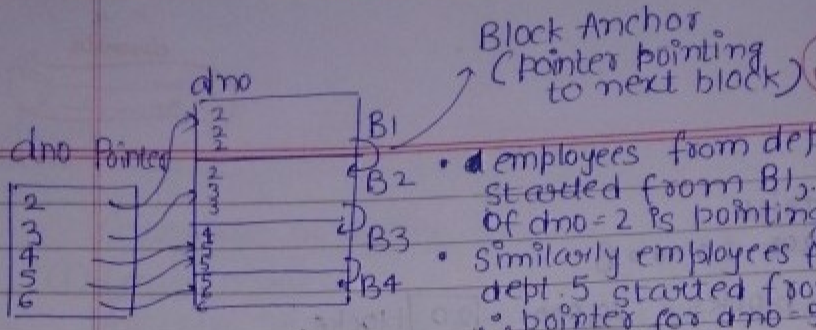
★ Atmost 1 primary index is possible.

2. Clustering Index:-

Conditions:-

(1) Ordered file &

(2) Search key is $\boxed{\text{non-key}}$.



- employees from dept.2 started from B1, ∴ pointer of dno=2 is pointing to B1.
- Similarly employees from dept.5 started from B3, ∴ pointer for dno=5 is pointing to B3.

☆ eg. The block anchors are used when we want to access employees of dno=2

☆ Clustered Index is always sparse index.

☆ Atmost 1 clustering index is possible.
(because

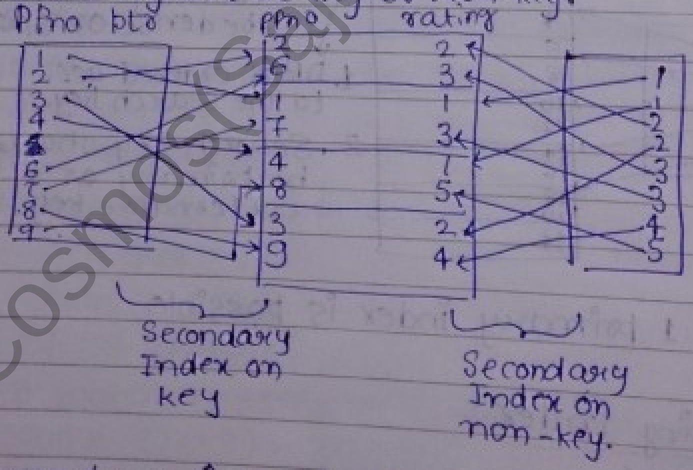
☆ either clustering index

Secondary Index :-

Conditions:-

(1) Unordered file.

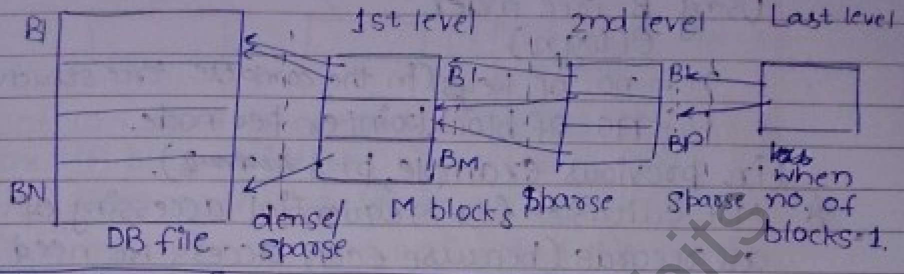
(2) Search key can be key or Non-key.



☆ Secondary index is always dense index
(because file is unordered.

☆ More than one secondary index is possible.

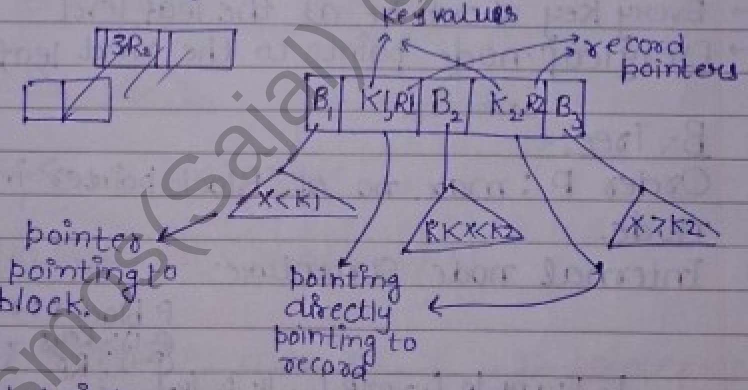
(4) Multilevel Index:-



$\boxed{\text{I/O Cost} = (\text{no. of levels}+1)\ \text{blocks}}$

Dynamic Multilevel Indexing

- B-Trees
- B^+-Tree Indexing } Balanced Search Tree Indexing



Key values
record pointers
B1
K1,R1
B2
K2,R2
B3
X<K1
RK<X<K2
X>K2
pointer pointing to block.
pointing directly pointing to record

1. Record pointer:-
pointer which points to data base.
(value pointer or data pointer).

2. Block pointer:-
pointer points to index block. (node pointer or tree pointer).

Worst Case I/O cost:

Using B-Tree index

$\Theta(\log_p n)$

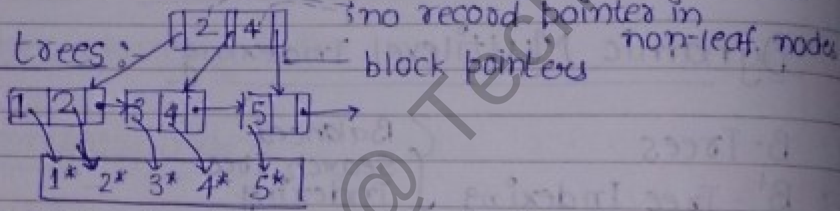
n :- no. of keys (in the complete tree structure).

p :- no. of blockpointer per node.

(in previous example, $p = 3$).

★ Not suitable for sequential accessing of all records (because each access we need to start from the root every time.)

B+ trees:



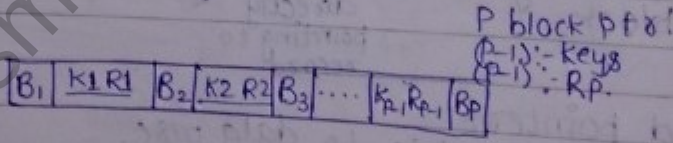
→ Every key should be at the leaf level

→ Every leaf node points to the next leaf node

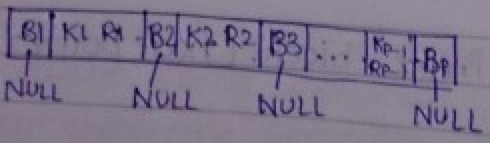
B-Tree:-

Order P : max. no. of Block pointer in B-Tree node.

① Internal node Structure:-

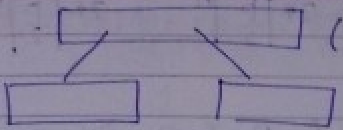


② Structure of leaf node:-



③ Every internal node except root should contain ~~be~~ atleast $\lceil P/2 \rceil$ block pointers & atmost P block pointers.

P = 15 (max 15 block ptr.)



(2 block ptrs to P block ptrs)

(⌈P/2⌉ block ptr. to max. PB$_P$)

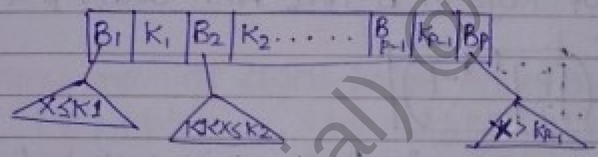
④ Root can ~~be~~ have atleast 2 block pointers & max. P block pointers

⑤ Every leaf node should be at same level & keys within the nodes should be in ascending order.

B$^+$ tree defn. :-

① Internal node structure
[P - block pointers
(P-1) = keys (no. of keys)]

②



② Structure of leaf node

| K1 R1 | K2 R2 | K3 R3 | ··· | Km Rm | → Block pointer.

★ B Tree Order P : max. no. of block pointers per node.

Height/level	Min no. of nodes	Min. no. of block pointers	Min no. of keys
0/1	1	$\boxed{2}$	1
1/2	$\boxed{2}$	$2*\lceil \frac{P}{2} \rceil$	$2*(\lceil \frac{P}{2} \rceil -1)$
2/3	$2*\lceil \frac{P}{2} \rceil$	$\lceil \frac{P}{2} \rceil * (2*\lceil \frac{P}{2} \rceil)$	$2 \times \lceil \frac{P}{2} \rceil * (\lceil \frac{P}{2} \rceil -1)$
3/4	$\lceil \frac{P}{2} \rceil * (2*\lceil \frac{P}{2} \rceil)$	$\lceil \frac{P}{2} \rceil * (\lceil \frac{P}{2} \rceil * 2 * \lceil \frac{P}{2} \rceil)$	$\lceil \frac{P}{2} \rceil * (2*\lceil \frac{P}{2} \rceil) * (\lceil \frac{P}{2} \rceil -1)$

$h \qquad 2*\lceil \frac{p}{2} \rceil^{h-1} \qquad 2*\lceil \frac{p}{2} \rceil^{h} \qquad 2*\lceil \frac{p}{2} \rceil^{h-1} * (\lceil \frac{p}{2} \rceil - 1)$

∴ So, min. no. of keys in B-Tree with order p & height h :-

$$1 + 2*(\lceil \tfrac{p}{2} \rceil - 1)\left[1 + \lceil \tfrac{p}{2} \rceil + \lceil \tfrac{p}{2} \rceil^2 + \dots + \lceil \tfrac{p}{2} \rceil^{h-1}\right]$$

$$= 1 + 2*\cancel{(\lceil \tfrac{p}{2} \rceil - 1)}\left(\frac{1(\lceil \frac{p}{2} \rceil^{h} - 1)}{\cancel{\lceil \frac{p}{2} \rceil - 1}}\right)$$

$$= \boxed{1 + 2*(\lceil \tfrac{p}{2} \rceil^{h} - 1)}$$

- Min. no. of keys in B-Tree with order p & level l :-

$$\boxed{1 + 2*(\lceil \tfrac{p}{2} \rceil^{l-1} - 1)}$$

- Min. no. of nodes in B-Tree with order p & height h :-

$$1 + 2 + 2*\lceil \tfrac{p}{2} \rceil + 2*\lceil \tfrac{p}{2} \rceil^2 + \dots + 2*\lceil \tfrac{p}{2} \rceil^{h-1}$$

$$= 1 + 2\left(1 + \lceil \tfrac{p}{2} \rceil + \dots + \lceil \tfrac{p}{2} \rceil^{h-1}\right)$$

$$\boxed{1 + 2*\left(\frac{1(\lceil \frac{p}{2} \rceil^{h} - 1)}{(\lceil \frac{p}{2} \rceil - 1)}\right)} = 1 + 2*\left(\frac{(\lceil p/2 \rceil^{l-1} - 1)}{(\lceil p/2 \rceil - 1)}\right)$$

-

Height	Max no of nodes	Max no. of block ptrs.	Max no. of keys.
0	1	p	$(p-1)$
1	p	$p\times p$	$p\times(p-1)$
2	p^2	p^3	$p^2\times(p-1)$
⋮			
h	p^h	p^{h+1}	$p^h\times(p-1)$

- Max. no. of keys in B-Tree with order P & height h.

$$(p-1)+p\times(p-1)+p^2\times(p-1)+\dots+p^h\times(p-1)$$
$$=(p-1)[1+p+p^2+\dots+p^h]$$
$$=\cancel{(p-1)}\left[\frac{\cancel{p}1(p^{h+1}-1)}{\cancel{(p-1)}}\right]$$
$$=\boxed{p^{h+1}-1}$$

- Max. no. of block point nodes in B-Tree :-

$$1+p+p^2+\dots+p^h=\boxed{\frac{1(p^{h+1}-1)}{(p-1)}}$$

- Height of B-Tree with order P & N keys.

$$N=1+2\left(\left\lceil\frac{p}{2}\right\rceil^h-1\right)$$ [taking min. keys]

$$\Rightarrow \frac{(N-1)}{2}+1=\left\lceil\frac{p}{2}\right\rceil^h$$

$$\Rightarrow \boxed{h=\log_{\lceil\frac{p}{2}\rceil}\left(\frac{N+1}{2}\right)}$$ max. height while taking (~~min~~) when taking min. keys.

$N=p^{h+1}-1$ (Each node consists max. possible keys.)

$h=\log_p(N+1)-1$ → min. height.

☆ If node occupancy is min., then no. of nodes are max., if node occupancy is max., no. of nodes becomes min.

Q. Identify min. & max. keys & nodes in B-Tree with order p=5 & level l=4.

Ans. Min.

level	nodes	keys	block pointers
1	1	1	2
2	2	2*2=4	2*3
3	~~2*~~6	6*2=12	6*3
4	18	18*2=36	18*3

min. no. of keys = 53
min. no. of nodes = 27

Max.

Level	nodes	block pointers	keys
1	1	5	4
2	5	5×5=25	5×4=20
3	25	25×5=125	25×4=100
4	125	125×5	125×4=500

max. no. of keys = 624
max. no. of nodes = 156

* B Tree Order P: max. no. of key's in B Tree Nodes (means p = the max. no. of keys that can be stored in a node.)

max. no. of keys & ~~order~~ nodes in B-Tree with order p=5 & level 4.

level	max. no. of nodes	max. no of BP	max. no. of keys
1	1	6	5
2	6	6^2	6×5
3	6^2	6^3	6^2×5
4	6^3	6^4	6^3×5

order P is defined ~~as~~ as: b/w P & 2P keys (for internal node)

~~as~~ b/w 1 & 2P keys (for root nodes)

using above order identify min., max. no. of keys & nodes in B-Tree with order 4 & level 4.

min.

level	~~ma~~ nodes	block pointers	keys
1	1	2	1
2	2	2×(P+1) = 10	2×P = 8
3	2×(P+1) = 10	10×5 = 50	10×4 2×(P+1)×P = 40
4	50	50×5 = 250	50×4 = 200

no. of nodes = 63
no. of keys = 249

max.

Level	nodes	block pointers	keys
1	1	9	8
2	9	9×9 = 81	9×8 = 72
3	81	81×9 = 729	81×8 = 648
4	729	729×9	729×8 =