DBMS [8-10 marks]

Syllabus:-

1. Integrity Constraints $$ ER Model — 2 marks
2. Normlization — 2-4 marks
3. Query (RA, SQL, RC) — 4 marks
4. File org and Indexing — 2-4 marks
5. Transactions and concurrency Control — 2-4 marks

- Notes
- WB| GATE
- Text Book exercise Problems
  - → DBMS (Raghuram krishna)
  - → DBMS (Navathe)

Introduction:

Data Base — Collection of related data
ex:- Set of student's info.

DBMS — Software which is used to manage data base files.

does not know how the files actually stored.

DBMS [8-10 marks]

Syllabus:-

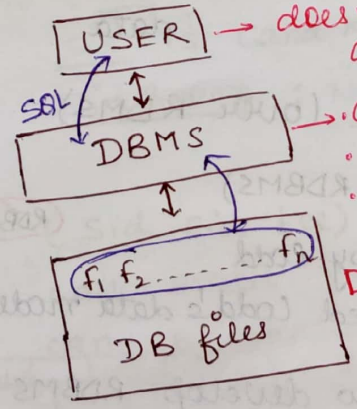
1. Integrity Constraints $$ ER Model — 2 marks
2. Normalization — 2-4 marks
3. Query (RA, SQL, RC) — 4 marks
4. File org and Indexing — 2-4 marks
5. Transactions and concurrency Control — 2-4 marks

- Notes
- WB|GATE
- Text Book exercise problems
  - → DBMS (Raghuram krishna)
  - → DBMS (Navathe)

Introduction:

Data Base — Collection of related data
ex:- Set of student's info.

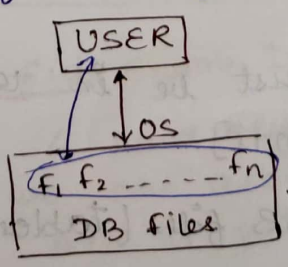
DBMS — Software which is used to manage data base files.



- does not know how the files are actually stored.
- acts as an interface
- it controls all files
- it is responsible for storing files
- it contains all info about how the data is stored.

data independency:
USER can be able to retrieve the data without knowing how it is actually stored.

Flat file system (OS files)
DB files manages by user without DBMS.



It is good for small database.
but for huge DB then it fails to manage the DB

User should know complete structure & data storage

user is responsible about how data is stored, i.e, File structure & data format is user responsibility.

Flat file system can be used to manage small DB. if DB is too huge, flat file system fails to manage DB

Limitation of flat file

⇒ Too complex to manage application programs and development of application programs

eg:- program (statements) → USER → DB file (disk) — it is complex writing program.

⇒ Requires more I/O cost to access data from DB files

⇒ Provides less degree of concurrency

Degree of concurrency
no. of users can use DB simultaneously.

⇒ Maintaining non redundant data is too complex almost impossible.

Advantage of DBMS file system

⇒ Because of data independency development of application programs becomes simple.

Data Independency:
Hiding storage info to the user.

⇒ Because of Indexing DBMS file system access cost is too less

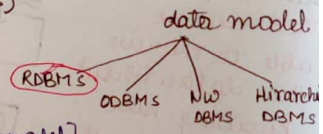
⇒ Because of concurrency control of DBMS, can possible more degree of concurrency.

⇒ uses normalization or schema refinement used to maintain non redundant data.

Integrity Constraints (over RDMS)

Relational DBMS (RDBMS)

- ~~produc~~ Proposed by Codd
  [also called Codd's data model]



- Codd 12 Rules to develop RDBMS S/W (software)
  (RDBMS guidelines)

RDBMS Guidelines

- data in DB files must be in tabular format.
  [Set of rows & columns]
- No two rows of DB files (tables) is same

NULL: Unknown/unexisted value

eid	Ename	DOB	PanID	Adhar no	Ifsc	Ano
e1	A	1995	X5		SB101	101
e2	A	NULL	NULL		CAN01	101
e3	B		X2		SB102	102
e4	B		NULL		CAN02	102

Not NULL: Attribute not allowed NULL's

Primary Key: Any one Cand Key of relational schema whose feild values always NOT NULL

Constraint:
- must be one of C_K
- feild values not allowed NULL's
- Almost one PK allowed for any Relational schema

Alternative Keys: All C_K of relational schema except ~~except~~ P_K

Constraint:
- must be one of C_K
- field value allow NULL
- many AK's are allowed

eg:-
```
CREATE TABLE Emp
(eid      Varchar(10)   Primary Key,
 ename    Varchar(30)   NOT NULL,
 PanID    Varchar(9)    UNIQUE,
 AdharNO  integer(12)   UNIQUE NOT NULL,
 Ifsc     Varchar(6),
 ano      integer(10),
 UNIQUE (ifsc, ano)
);
```

Simple Candidate Key:
Candidate Key with only one attribute
eg:- eid, panID, Adharno

Compound Candidate Key:
Candidate Key with at least two attributes
eg:- Ifsc Ano

Prime Attribute (KEY attribute)
Attribute which belongs to some candidate Key of relational scheme.
eg:- eid, panid, Adharno, Ifsc, Ano

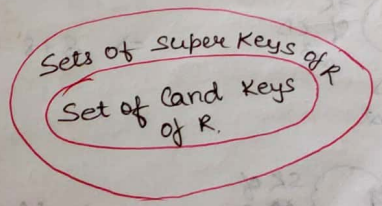
Prime Attribute set of Emp {eid, panID, Adharno, Ifsc, Ano}

Non-Prime Attribute set of Emp {ename, DOB}

Super Key (used for DB design)
Set of attributes which can differentiate records of relational schema uniquely.
(may not minimal attribute set)
Constraint:
field values unique for all the records.

eg:- Student (Sid, Sname DOB)
Cand Key: {Sid}
Super Keys: { Sid ← minimal S.K
Sid Sname
Sid DOB
Sid Sname DOB
}

Candidate Key:
minimal Super Key



Q- R(ABCD)

How many SK's in R if {A} cand Key ?

SK <=> [C_K] . [any subset of other attributes]

A. { Any subset of (BCD) } 2^3

{ A ABC
AB ABD
AC ACD
AD ABCD }

8 super keys

Q- R(ABCDE)

How many S_K in R if

(i) {AB, BC} C_K (ii) {AB, CD} Cand Key

(iii) {A, BC} C_K

(i) AB = {CDE} = 2^3
BC = {DE} = 2^2
12

(ii) AB = {CDE} = 2^3
CD = {E} = 2^1
10 ✗

(iii) A = {BCDE} = 2^4
BC = {DE} = 2^2
20

(OR)

(i) SK's of CK AB 2^3 — SK of CK BC 2^3 — SK of CK ABC = 2^2

n(X∪Y) = n(X) + n(Y) - n(X∩Y)
⇒ 8+8 -4
= 16-4 = 12

(ii) SK of AB 2^3 — SK of CD 2^3 — ABCD 2^2

⇒ 8+8 -2 = 14

(iii) SK of A 2^4 — SK of BC 2^3 — ABC 2^2

⇒ 16+8-4
= 20

Q- R(ABCDEF)

How many S_K in R if

(i) {A, B, C} Cand Keys (ii) {A, BC, CD} C_K GATE

(iii) {AB, BCD, CDE} C_K

(i) AB = 2^4, A 2^5, B 2^5, C 2^4, AC 2^4, 2^3

⇒ 32+32+32 -16 -16 -16 +8
= 96 -48 +8
= 56

(ii) 2^3, A 2^5, BC 2^4, 2^3, 2^3, CD 2^4, ABCD 2^2

⇒ 32+16+16-8-8-8+4
= 64-24+4
= 44

(iii) AB 2^4, 2^2, BCD 2^3, ABCDE = 2, BCDE 2^2, 2, CDE 2^3

= 16+8+8-2-4-4+2
= 32-8
= 24

W, Z, X, Y

⇒ (+) → Individual, 3 sets, 5 sets
(-) → 2 sets, 4 sets

Q- R(ABCDE)

{AB, BC, CD, DE} Cand keys

Super Key { AB ABD BC CD DE }
ABC ABE BCD ACD BDE
ABCD ABCE BCDE CDE ADE
ABCDE ABDE BCE ACDE

19 SK's

Q- $R(A_1 A_2 A_3 \ldots A_n)$

How many S-K in R?

(a) $n!$ (c) $2^n - 1$

(b) 2^n (d) 2^{2^n}

$R(\underline{A_1}\ \underline{A_2}\ \underline{A_3})$

{ A1, A2, A3, A1A2, A1A3, A2A3, A1A2A3 } → max SK's $2^3 - 1 = 7$

$2^n - 1$ max of S-K's if each attribute of R is C-K

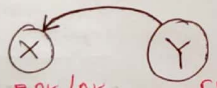
(C)

Foreign Key (Referencial Key)

⇒ used to relate data b/w tables

⇒ defined over two tables
Referenced Relation
Referencing Relation

⇒ def: FK is set of attributes references to primary key / Alternative key of same relation / some other relation



eg: Stud (Sid, Sname, DOB) Enroll (Sid, Cid, fee)

Stud Sid	Sname	DOB
S1	A	1995
S20	A	1995
S3	B	1994
S4	B	1994
S5	C	1994
S6	D	1996

Enroll Sid	Cid	fee
S1	C1	5000
S1	C2	6000
S2	C2	4000
S4	C1	4000
S10	C1	5000

Referenced Relation; Referencing relation

D → I → D → U

(1 to many)

restricted because, it is not in stud relation data is inconsistent

eg: Emp (eid, ename, SupID) also must be emp it ON delete Cascade

	eid	ename	SupID
del	e1	A	NULL
	e2	B	e1
	e3	B	e1
	e4	C	e2
	e5	A	NULL
	e6	D	e5 ✓ e8 → it is not an emp

#Cascading delete = 3

Total delete = 4

eg: R (PK A, AK B, C) S (D, FK E)

R A	B	C
a1	2	c1
a2	4	c2 → D
a3	null	c2
a4	6	c3
a5	null	c1

S D	E
2	e1
2	e2
(null)	e3 → it is not related to any reference record.
null ✓ 4	e4
6 ✓ 10	e5

non-relation is allowed in both the sides

D can be NULL or values of B

not allowed are

Foreign key allows NULL value ~~and~~ referencing record whose foreign key value is null, not related to any referenced record (out of referential integrity constraint)

each record of Referenced Relation can related many referencing record and each record of referencing relation can relate atmost one referenced record.

Referential Key Integrity Constraint (FK IC):

(IC)

(1) Referenced Rel (Stud)

(a) Insertion: NO voilation

(b) Deletion: may cause voilation

default (i) ON DELETE NO ACTION

Deletion of Referenced Record is not allowed if FK voilation occurs

(ii) ON DELETE CASCADE

Deletion of Referenced Record is allowed and DBMS deletes all related referencing records.

(iii) ON DELETE SET NULL

If FK field allowed to set NULL, set NULL in FK values and allowed referenced record.

(c) Updation : may cause violation

default (i) ON UPDATE no Action

(ii) ON UPDATE CASCADE

(iii) ON UPDATE SET NULL

(II) Referencing Rel (ENROLL)

(a) Insertion : may cause violation

Restricted

(b) Deletion : NO violation

Open

(c) Updating : may cause violation

Insertion, Updation of referencing Rel is restricted if F.K violation occurs.

eg:- Stud (Sid Sname DOB)

Enroll (Sid Cid fee)

```
CREATE TABLE ENROLL
( Sid    varchar(10),
  Cid    varchar(10),
  fee    integer,
  primary key (sid,cid),
  FOREIGN KEY (Sid)
  REFERENCES Stud(Sid)
  ON DELETE CASCADE
  ON UPDATE CASCADE
);
```

it is present when foreign key is present

if not mentioned then references to it's own table

it act W.R.T stud relation

NORMALIZATION (Schema Refinement)

normalization used to reduce or eliminate redundancy in relational schema.

If two or more independent relations stored in single relation which can cause redundancy

eg:-

Sid → Sname DOB (Stud info)

Cid → Cname Instructor (Course info)

Sid Cid → fee (mapping)

R

Sid	Sname	DOB	Cid	Cname	Ins^r	fee
S1	A	1995	C1	DB	Koath	
S2	A	1995	C1	DB	Koath	
S3	B	1995 1994	C1	DB	Koath	
S3	B	1994	C2	OS	Galwin	
S3	B	1994	C3	Algo	Corman	

C4

Problems because of Redundancy

[DB Anamalies]

(a) Insertion Anamaly

to insert some data there must be other independent data.

eg:- to insert Course, there must be some student enroll for course.

(b) Deletion Anamaly

Deletion of some data may lost some other independent data

eg:- because of deletion of student may lost course info.

(c) Updation Anamaly

If some redundant copy may updates, other redundant copies failed to update, it cause inconsistency

Normalization of DB:

decompose relation into two or more sub relation to reduce / eliminate redundancy and DB anamalies.

eg: R_1 (Sid Sname DOB) R_2 (Sid Cid fee) R_3 (Cid Cname In

R_1			R_2			R_3		
S1			S1	C1	–	C1		
S2			S2	C1	–	C2	OS	G
~~S3~~			~~S3~~	~~C1~~	–	C3	Algo	Co
			~~S3~~	~~C2~~	–	C4	CO	Hay
			~~S3~~	~~C3~~	–			

NO Redundancy
NO DB Anamalies
> Goals of Normalized DB design

Functional Dependency (FD)

X, Y are some set of Attributes over R

eg: R (X Y) $X \longrightarrow Y$

$t_1 \to$	x_1	y_5
t	x_2	y_2
	x_3	y_2
	x_4	y_4
	x_1	y_5
$t_2 \to$	x_1	y_5

[Y determines by X]

$X \longrightarrow Y$ FD implied in R

if $t_1 \cdot X = t_2 \cdot X$ then $t_1 \cdot Y = t_2 \cdot Y$

eg: R (Sid Sname Cid)

S1	A	C1
S1	A	C2
S2	B	C2
S3	B	C3
S4	C	C3

{ Sid → Sname
Sid Cid → Sname }

eg:- R (X Y) $X \longrightarrow Y$ not implied in R

$t_1 \to$	x_1 y_1
$t_2 \to$	x_2 y_2

$\because$ $t_1 \cdot X = t_2 \cdot X$ but $t_1 \cdot Y \neq t_2 \cdot Y$

Trivial FD (Reflexive FD)

$X \longrightarrow Y$ FD is trivial iff $X \supseteq Y$

attribute determines themselves

eg:
- Sid → Sid
- Sname → Sname
- Cid → ~~Cname~~ Cid
- Sid Sname → Sname
- Sid Sname → Sid Sname

] trivial FD

Non Trivial FD

$X \longrightarrow Y$ non trivial FD if no common attribute in X, Y attribute sets.

attribute determine some other attribute

eg:
- Sid → Sname
- Sid Cid → Sname

Semi Non Trivial FD

Combination of trivial & non trivial FD

eg:
- Sid → Sid Sname
- Sid Cid → Cid Sname
- Sid → Sid
- Sid → Sname
- Sid Cid → Cid
- Sid Cid → Sname

(if $X \longrightarrow YZ$ then $X \longrightarrow Y$, $X \longrightarrow Z$)

Every possible trivial FD over attributes of relation R is always implied on relation R

Armstrong Rules Over FD's

1. Reflexivity: $X \longrightarrow X$ always in relation R (trivial FD)
2. Transitivity: if $X \longrightarrow Y$, $Y \longrightarrow Z$ then $X \longrightarrow Z$
3. Augmentation: if $X \longrightarrow Y$ then $XZ \longrightarrow YZ$ always true

eg:- Sid ⟶ Sname

⇒ Sid Cid ⟶ Sname Cid

eg: A ⟶ B ⇒ AC ⟶ BC

AC ⟶ BC ⇏ A ⟶ B

4. Split Rule :

if X ⟶ YZ then,

X ⟶ Y

X ⟶ Z

5. Merge Rule :

if X ⟶ Y , X ⟶ Z then,

X ⟶ YZ

eg:- {A ⟶ B} ⇒ AC ⟶ B (split rule)

⇓

AC ⟶ BC (aug)

Attribute Closure (x^+)

x^+ = { Set of all attributes that can determines by attribute x }

eg:- Given FD Set

{ A ⟶ B , C ⟶ D , AB ⟶ E , BE ⟶ C , EF ⟶ G }

A^+ = { A, B, E, C, D } , A ⟶ ABCDE

↓ reflexivity

$(AF)^+$ = { ABCDEFG } , AF ⟶ ABCDEFG

Super Key

eg: R(A B C D) { ..... } FD

2

3

4

5

5 7 8

A^+ = { ABCD }

if A^+ = { ABC } then A is not S.K

x is some attribute set over R

x is super key of R

iff x^+ must determine all attribute of Rel R

Candidate Key (minimal S.K)

x is candidate key of R

iff ① x must be S.K of R

x^+ = { All attribute of R }

(and)

② No Proper subset of X is S.K of R

∀ Y ⊂ X such that

Y^+ : not determines all attribute of R

eg:- R(ABCDE)

{ AB ⟶ C, C ⟶ D, B ⟶ E }

$(AB)^+$ = { A B C D E } ∴ AB is S.K

A^+ = { A } , B^+ = { B E } → not S.K.

∴ AB is C.K of relational schema.

Q- Find C.K of given Rel R with FD set :

1. R(A B C D E F) { F ⟶ A , D ⟶ E , C ⟶ F }

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

↑ min S.K

∴ BCD is C.K

X ⟶ Y

F ⟶ A

D ⟶ E

C ⟶ F

F^+ = F

C^+ = C

D^+ = D

2. R(A B C D E F) { B ⟶ D, C ⟶ A, E ⟶ B }

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

↑ min SK ∴ CEF is C.K

3. R(ABCDEF)

{AB→C, C→D, CD→E, DE→F, EF→B}

$(AB)^+$ = ABCDEF

$(AC)^+$ = ACDEFB

$(AEF)^+$ = AEFBCD // $AE^+ = AE$, $AF^+ = AF$

$(AED)^+$ = ABCDEF

$(A\cancel{CD})^+$ $(\cancel{AE})^+$

4. R(ABCDE) {AB→C, C→D, CD→E, DE→B}

Prime, Prime, Prime, Prime

∴ more than 1 C.K exist

** must belong to all C.K

$(AB)^+$ = {ABCDE} // find one C.K

$(AD)^+$ = {ACDEB}

{Non trivial FD X→(Y) Prime.} ⟸ {Rel atleast two candidate in R}

$(ADE)^+$ = {ADEBC} // $AD^+ = AD$, $AE^+ = AE$

$(ACD)^+$ = {ACDEB} // AC^+ = {ACDEB}

(not minimal S.K)

{AB, ADE, AC}

5. R(ABCDEF)

{AB→C, C→DE, E→F, F→B}

Prime, Prime, Prime

{AB, AF, AE, AC}

$(AB)^+$ = {ABCDEF}

$(AF)^+$ = {AFBCDE}

$(AE)^+$ = {AEFBCD}

$(AC)^+$ = {ACDEFB}

6. R(ABCD)

{AB→CD, C→A, D→B}

{CD, AB, AD, CB}

$(CD)^+$ = {CDAB}

$(AB)^+$ = {ABCD}

$(AD)^+$ = {ADBC}

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

$(CB)^+$ = {CBAD}

7. R(ABCDEFGH)

{DA, DE, DF, DB}

{CH→G, A→BC, B→CFH, E→A, F→EG}

$(DA)^+$ = DABCFHEG

$(DE)^+$ = DEABCFGH

$(DF)^+$ = DFEGABCH

$(DB)^+$ = DBCFHAGE

$(\cancel{DCH})^+$ = DCHG

$(\cancel{DG})^+$ = DG

④

Finding C.K of Relational schema with n attribute is NP complete problem (exponential TC)

$O(2^n)$

Membership Test

X→Y, FD member of FD set (F)

iff x^+ must determine Y in FD set (F)

F = {....} (X→Y) member of F

x^+ = {....Y....}

ex: {AB→C, BC→D, CD→E, E→F}

(i) AB→F (ii) ABG→E (iii) BC→A

$(AB)^+$ = {ABCDEF} Yes

$(ABG)^+$ = {ABGCDEF} Yes

$(BC)^+$ = {BCDEF} Not member of FD set

Equality Test for FD set :

F = {.....} G = {.....}

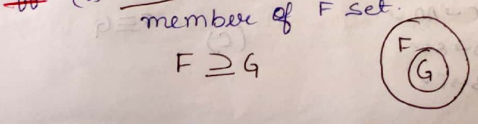
X→Y, x^+ = {...Y...}

A→B, A^+ = {...B...}

Given FD sets F and G logically equal

iff (i) F cover G : Every FD of G set must member of F set.

F ⊇ G



and (ii) G covers F : Every FD of F set must member of G set.

$F \subseteq G$

eg:- F covers G G covers F

T	T	⇒	F ≡ G
T	F	⇒	F ⊃ G
F	T	⇒	F ⊂ G
F	F	⇒	F $ G

not comparable

F ∩ G disjoint

eg:- $F = \{A \to B, B \to C, C \to A\}$

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

which is true ?

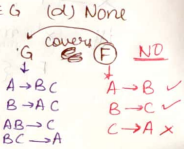
(a) F ⊂ G (b) F ⊃ G (c) F ≡ G (d) None

G ⊂ F

$A^+ = \{ABC\}$ $A \to B$ ✓

$B^+ = \{BAC\}$ $B \to C$ ✓

$C^+ = \{C\}$ $C \to A$ ✗



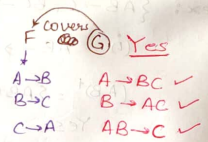
BC ⊄ FC G

$A^+ = \{ABC\}$ $A \to BC$ ✓

$B^+ = \{BCA\}$ $B \to AC$ ✓

$C^+ = \{CAB\}$

(b)



eg:- $F = \{A \to BCDEF, BC \to ADEF, AC \to E, D \to E, B \to F, AB \to F\}$

$G = \{A \to BC, BC \to AD, D \to E, B \to F\}$

F covers G Yes

$A \to BCDEF$ ✓
$BC \to ADEF$ ✓
$AC \to E$ ✓
$D \to E$ ✓
$B \to F$ ✓
$AB \to F$ ✓

$A \to BC$ ✓
$BC \to AD$ ✓
$D \to E$ ✓
$B \to F$ ✓

G covers F Yes

(c)

$F \equiv G$

Properties Of Decomposition :

1. Lossless Join Decomposition
2. Dependency Preserving Decomposition

↳ Lossless Join Decomposition

M : Natural Join

Relational schema R with instance (r) decomposed into R1, R2, ... sub relations

In General,

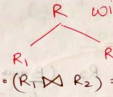
$(R_1 \bowtie R_2 \bowtie R_3 \ldots \bowtie R_n) \supseteq r$

if $(R_1 \bowtie R_2 \bowtie \ldots \bowtie R_n) = r$

then decomposition is lossless join

if $(R_1 \bowtie R_2 \bowtie \ldots \bowtie R_n) \supset r$

then lossy join decomposition



1. $(R_1 \bowtie R_2) = r$ LLJ
2. $(R_1 \bowtie R_2) \supset r$ lossy join
3. $(R_1 \bowtie R_2) \subset r$ Not possible

eg: R (Sid Sname Cid) {Sid → Sname}

C_K : Sid Cid

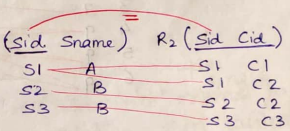
r

Sid	Sname	Cid
S1	A	C1
S1	A	C2
S2	B	C2
S3	B	C3

① decomposed into R_1 (Sid Sname) R_2 (Sid Cid)

$R_1 \bowtie R_2 = r$

Sid	Sname	Cid
S1	A	C1
S1	A	C2
S2	B	C2
S3	B	C3



Lossless Join decomposition

② decomposed into $R_1 \bowtie R_2 \supseteq R$

Sid	Sname	Cid
S1	A	C1
S1	A	C2
S2	B	C2
S2	B	C3
S3	B	C2
S3	B	C3

R_1 (Sid Sname) R_2 (Sname Cid)

Sid	Sname
S1	A
S2	B
S3	B

Sname	Cid
A	C1
A	C2
B	C2
B	C3

∴ it is lossy join

Spurious records (records which are not present in original instances) so originality of original data is lost.

NOTE: Because of spurious Records, data access using SQL Query causes inconsistency. (Result of SQL Queries is incorrect)

R(A B C) {A → B, B → C}

A : C_K

R

2	4	7
3	4	7
4	5	9
5	5	9
6	6	9

ⓐ R_1 (A B)

A	B
2	4
3	4
4	5
5	5
6	6

R_2 (B C)

B	C
4	7
5	9
6	9

lossless

ⓑ R_1 (A C)

A	C
2	7
3	7
4	9
5	9
6	9

R_2 (B C)

B	C
4	7
5	9
6	9

lossy

Lossless : Common attribute of any of the sub relation is CK or SK (Unique)

Lossy : Common attribute is not SK for atleast one subrelation (not Unique)

$R_1 \cap R_2 = (D^+) = \{DIJ\}$

S.K for R_1

$R_1 \bowtie R_2 = ADEIJ$

$\therefore (\overset{R_1}{ADEIJ}, \overset{R_2}{FGH}, \overset{R_3}{ABCF})$

$R_2 \cap R_3 = (F)^+ = \{FGH\}$

S.K for R_2

$(\overset{R_1}{ADEIJ}, \overset{R_2}{ABCFGH})$

$R_1 \cap R_2 = A^+ = \{ADEIJ\}$

S.K for R_1

$\therefore (R \; ABCDEFGHIJ) = R$

Lossless

(ii) $\{\overset{R_1}{FGH}, \overset{R_2}{DIJ}, \overset{R_3}{ADEBF}, \overset{R_4}{ABC}\}$

$R_2 \cap R_3 = D^+ = \{DIJ\}$

S.K for R_2

$\{\overset{R_1}{FGH}, \overset{R_2}{ABDEFIJ}, \overset{R_3}{ABC}\}$

$R_2 \cap R_3 = (AB)^+ = \{ABC\}$

S.K for R_3

$\{FGH, ABCDEFIJ\}$

$R_1 \cap R_2 = F^+ = \{FGH\}$

S.K for R_1

$\therefore$ Lossless.

Dependency Preserving Decomposition

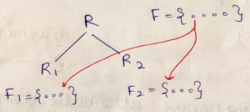
Relational schema R with FD set (F) decomposed into $R1, R2, \ldots, Rn$ sub relations with FD sets $F_1 F_2 \ldots F_n$.

In General,

$\{F_1 \cup F_2 \cup \ldots \cup F_n\} \subseteq F$

if $\{F_1 \cup F_2 \cup \ldots \cup F_n\} = F$ then DP decomposition

if $\{F_1 \cup F_2 \cup \ldots \cup F_n\} \subset F$ then not DP decomposition



1. $\{F_1 \cup F_2\} = F$ — DP decomp
2. $\{F_1 \cup F_2\} \subset F$ — Not DP decomp
   $\{F_1 \cup F_2\} \supset F$ — Not possible

eg: R (ABCDE) with FD set $A \to B, B \to C, C \to D, C \to B, D \to BE$

decomposed into $\{AB, BC, CD, DE\}$

R_1 (A B)	R_2 (B C)	R_3 (C D)	R_4 (D E)
$A \to B$ (F_1)	$B \to C$, $C \to B$ (F_2)	$C \to D$, $D \to C$ (F_3)	$D \to E$ (F_4)

$A \to B$ ✓
$B \to C$ ✓
$C \to D$ ✓
$D \to BE$ ✓

$\{F_1 \cup F_2 \cup F_3 \cup F_4\} = F$

$\therefore$ DP decomposition

Q- $R(ABCD)$ $\{AB \to CD, D \to A\}$ decomposed into $\{ABC, AD, BCD\}$

R_1 (A B C)	R_2 (A D)	R_3 (B C D)
$AB \to C$	$D \to A$	$BD \to C$

$AB \to CD$ ✗

$(BD)^+ = \{BDAC\}$

$\therefore$ Not DP decomposition.

Q- R(ABCDEF) {AB→C, C→D, D→E, ...}

{BCD, AB, DE, EF}

R_1(BCD)	R_2(AB)	R_3(DE)	R_4(EF)
BC → D		D → E	E → F
BD → C			
C → D			

AB → C ×
F → A ×

Not DP decomp.

Canonical Cover [Minimal Cover]

minimal set of FD's which are logically equal to given FD set F

Given FD set (F)

determinant (X) → Y

Procedure :

1. Remove extraneous attribute from determinate of each FD (unnecessary)

eg: {XYZ → W , X^+ = {... Y ...}}

Y extraneous, X → Y

Q- find the extraneous attribute

(i) {ABCD → E, A → F, F → D}

D ≡ A → D ⇒ {ABC → E, A → F, F → D}

(ii) {AB → C, A → B}

⇒ {A → C, A → B} B

2. Remove redundant FD's from step (1) result

X → Y FD of FD set (F) is redundant iff X → Y must member of {F − (X → Y)}

eg: {A → B, B → C, (A → C)}

× Redudant because it can be derived from other FD

Q- {AB → C, (AB → D), BC → D}

×

AB → CD, BC → D

$(AB)^+$ = ABCD

eg:- {ABCD → E, BC → D, A → F, F → C, AB → F}

(1) ABD → E, BC → D, A → F, F → C, AB → F ×

Removal of Extraneous

A BC D → E
BC → D
A → F
F → C
AB → F

Redundant

AB → E
BC → D
A → F
F → C
AB → F ×

AB^+ = ABE CD

∴ minimal cover = {AB → E, BC → D, A → F, F → C}

Q- {A → BC ×
D → E ×
E → C
D → AEH
ABH → BD
DH → BC
}

D → A ✓
D → E ✓
D → H ✓
ABH → B ×
ABH → D ✓
DH → B
DH → C ×

A → BC ✓
D → E ×
E → C ✓
D → A ✓
E ✓
H ✓
AH → B ×
D ✓
D → B ×
C ×

Result:

D → AEH ✓
ABH → D ✓
E → C ✓
A → BC ✓

Minimal set of FD cover F may not unique but all minimal cover are logically equal which is also equal to the FD set F

eg:- F = { AB → C
A → B
B → A
}

AB → C: A → C, A → B, B → A ⇓ F_1 { A → BC, B → A, A → B }

AB → C: B → C, A → B, B → A ⇓ F_2 { A → B, B → AC, B → A }

$F_1 = F_2 = F$

Normal Forms

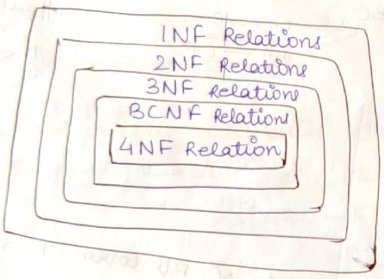
used to identify degree of redundancy

Over FD ($X \to Y$): 1NF, 2NF, 3NF, BCNF (0% redundancy over FD's)

Over MVD ($X \to\to Y$): 4NF (0% redundancy over MVD's)

Redundancy

- Over non-trivial FD $X \to Y$ [Single valued Dependency]
  ⇒ To eliminate redundancy over FD's relation should decomposed into BCNF
- Over non-trivial MVD $X \to\to Y$ [multi valued Dep]
  ⇒ relation should decomposed into 4NF



First Normal Form (1NF)

⇒ Default normal form of RDBMS rel.

⇒ Relational schema are in 1NF

iff No multivalued attribute in R

[every attribute of R must be atomic / single valued]

⇒ R

eg:

Sid	Sname	Cid
S1	A	C1 C2
S2	B	C2 C3
S3	B	C3

MVA (multi valued attribute)

{Sid → Sname}

Not in 1NF

Not in RDBMS

1NF design ⟺ design C_K

R

Sid	Sname	Cid
S1	A	C1
S1	A	C2
S2	B	C2
S2	B	C3
S3	B	C3

SidCid : C_K

{Sid → Sname}

not S_K

R is in 1NF

eg: R(Sid Sname DOB Cid Phno email)

S1 A 1995 C1/C2 (2) P1/P2/P3 (*3) e1/e2/e3/e4 (*4) — not in 1NF

S2 B 1994 C2/C3 (2) P3/P4 (*2) e4/e5 (*2)

C_K : SidCidPhnoemail

Sid → Sname DOB (not S_K)

Sid	Sname	DOB	Cid	Phno	email
S1	A	1995	C1	P1	e1
S1	A	1995			
⋮	⋮				
S1	A	1995	C2	P3	e4
S2	B	1994	C2	P3	e4
⋮	⋮				
S2	B	1994	C3	P4	e5

24 tuples required

8 tuples required

**Degree of redundancy is very high in 1NF

it is in 1NF

1NF — default NF of RDBMS

2NF 3NF BCNF — reduce/eliminate redundancy over FD's

$X \to Y$ FD of R forms redundancy iff

1] Non Trivial FD

&&

2] X is not S_K

$X \rightarrow Y$ FD of R not forms redundancy

iff ① Trivial FD ($X \supseteq Y$)

(OR)

② X is S_K

eg: Emp (eid, ename, rating, hourlywage)

e1		10	500
e2		10	500
e3		10	500
e4		12	800
e5		12	800

S_K: eid → ename rating

rating → hourlywage (not S_K) forms redundancy

eg:- R(ABC(D EF)) (D, EF: non prime)

{AB, BC} cand key

{ AB → C, B → D, D → E, AD → F, C → A } (B → D, D → E, C → A: not S_K) ⇒ forms redundancy

FD $X \rightarrow Y$ forms redundancy [X: not S_K]

	1NF	2NF	3NF	BCNF
1. Proper subset of C_K → Non prime (PA → NPA)	✓	X	X	X
2. Non Prime → Non Prime (NP → NP)	✓	✓	X	X
3. Proper subset of C_K and Non Prime → Non Prime (P_NP → NP)	✓	✓	X	X
4. Proper subset of C_K → Proper subset of C_K (PA → PA)	✓	✓	✓	X

more redundancy

0% redundancy over FD's

Second Normal Form (2NF)

Relational schema R in 2NF

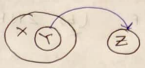
iff NO Partial dependencies in Rel R

Partial Dependency:

X: any C_K of R

Y ⊂ X

Z non prime of R



Y → Z : Partial dependency.

Third Normal Form (3NF)

Relational schema R in 3NF

iff every non trivial FD $X \rightarrow Y$ with

(i) X must be S_K

(Or)

(ii) Y must be prime Attribute

$X \rightarrow Y$: S_K (Or) Prime

{PA → PA} allowed in 3NF which can form redundancy.

Boyce Codd NF (BCNF)

Relational schema are in BCNF

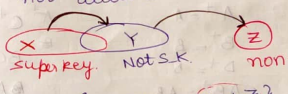
iff every non trivial FD $X \rightarrow Y$ with

(i) X must be S_K

$X \rightarrow Y$: S_K

(R in BCNF) <=> (0% redundancy in R over FD's)

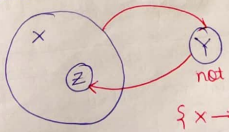
⇒ Non Prime Attribute transitively determined by S_K is not allowed in ~~BCNF~~ 3NF



{X → Y, Y → Z} not in 3NF

⇒ Prime Attribute transitively determined by superkey is allowed in 3NF but not allowed in BCNF

X: only C_K.



{X → Y, Y → Z} (Z: prime)

Q- R(ABCDE)

{ABD → C, BC → D, CD → E}

Highest NF satisfied by R?

(a) 1NF ✓ (b) 2NF ✗ (c) 3NF ✗ (d) BCNF ✗

ABD → C
BC → D
CD → E PD

$(ABD)^+ = ABCDE$

$(ABC)^+ = ABCDE$

{ABD, ABC} : C-K

{A, B, C, D} : PA

BCNF :-

ABD → C ✓
not S.K. BC → D ✗

3NF :-

ABD → C ✓ (S.K.)
BC → D ✓ (PA → PA)
CD → E ✗ (not S.K.) → NPA

2NF :-

ABD → C ✓ (S.K.)
BC → D ✓ (PA → PA)
CD → E ✗ (rel. E PA, PD) → NPA

2NF Testing :- $(\text{Proper subset of C-K})^+ = \{\text{only PA}\}$

Candidate key : ABD

$A^+ = A$
$B^+ = B$
$D^+ = D$
$AB^+ = AB$
$BD^+ = BD$
$AD^+ = AD$

ABC

$A^+ = A$
$C^+ = C$
$B^+ = BCDE$ (E non prime)
$AC^+ = AC$

∴ BC → E is PD in rel R

Q- R(ABCD)

{AB → C, BC → D}

BCNF ✗ 3NF ✗ 2NF ✓

$(AB)^+ = ABCD$ (↳ C-K)

AB ⇒ $B^+ = B$, $A^+ = A$ NO PD

Q- R(ABCD)

{AB → C, C → A, AC → D}

BCNF ✗ 3NF ✓ 2NF ✓ 1NF

$(AB)^+ = ABCD$
$(CB)^+ = ABCD$

A, C, B ← PA

AB ⇒ $A^+ = A$, $B^+ = B$

BC ⇒ $C^+ = ACD$

Q- R(ABCD)

{A → B, B → AC, C → D}

BCNF ✗ 3NF ✗ 2NF ✓

$A^+ = ABCD$
$B^+ = ABCD$

Proper subset of C-K

$B^+ = \phi$
$A^+ = \phi$

Q- R(ABCDEF)

{AB → C, C → D, CD → AE, DE → F, EF → B}

BCNF ✗ 3NF ✓

$(AB)^+ = ABCDEF$ ✓
$(AEF)^+ = ABCDEF$ ✓
$(AC)^+ = AE$
$(A)^+ = AF$
$(ADE)^+ = ABDEFC$ ✓
$(ACE)^+ = ACEDFB$

PA = A, B, C, D, E, F

Q- R(ABCDEF)

{AB → C, C → DE, E → F, F → B}

(AB, AF, AE, AC)

BCNF ✗ 3NF ✗ 2NF ✗ 1NF ✓

A, B, C, E, F

Q- R(ABCDE) $(AE)^+ = ABCDE$, $(DE)^+ = AE$, $(CE)^+ = CED$, $(BE)^+ = BCE$

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

BCNF

3NF

PA: A, B, C, D, E

Q A Relational schema R with only ~~or~~ simple C_K, no compound keys in relational schema

(a) R in 1NF but may not 2NF

~~(b)~~ R in 2NF but may not 3NF

(c) R in 3NF but may not BCNF

(d) BCNF

if C_K are simple then PD not possible

Q If Relational schema R with only Prime attribut then R always in 3NF but may not BCNF

(C)

Q Relational schema R with no non-trivial FD's are always in BCNF may not in 4NF

R(ABC) {No non trivial FD's}

ABC : C_K

NO non trivial FD's in R ⇒ No Redundancy in R over FD's (R in BCNF)

Q Relational schema R with only two attributes R always in

R(AB) $\{A \to B\}$ (A C_K) BCNF, also 4NF

R(AB) $\{B \to A\}$ (B C_K)

R(AB) $\{A \to B, B \to A\}$ (A C_K, B C_K)

Q- R (A B C D)

{A → B , C → D}

How many min relations required for 2NF

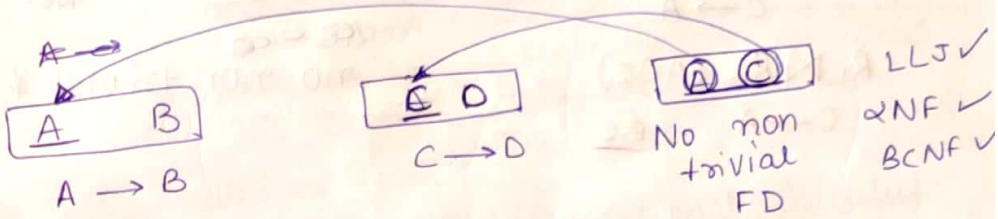
(a) 1 (b) 2 (c) 3 (d) 4

$(AC)^+ = ABCD$

$A^+ = AB$

$C^+ = CD$

(C)



Q R (A B C D E)

{AB → C , BC → A , AC → B} decompose into BCNF.

$(ABDE)^+ = ABCDE$

$(BCDE)^+ = ABCDE$

$(ACDE)^+ = ABCDE$

C_K : {ABDE , BCDE , ACDE}

AB → C BC → A AC → B

3NF but not BCNF

BCNF decomp:

R_2 (A B C)

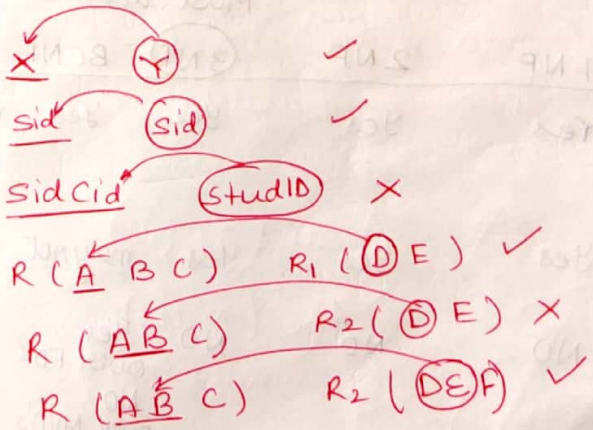
AB → C
BC → A
AC → B

AB BC AC

R_1 (AB DE) or BC or AC

no non trivial FD

LLJ ✓
DP ✓
BCNF ✓



5 NF [Join dependency NF] (JDNF)

If decomposition satisfy 4NF and lossless join decomposition, then decomposition is 5NF.

6 NF [Domain Key NF] (DKNF)

If 4NF, lossless join and DP of FD & MVD's is satisfied.

Queries

Procedural Query Lang

⇒ Formulation of how data retrieved from DB tabels

What data retrived from DB table

⇒ Relational Algebra

Non Procedural Query Lang

⇒ Formulation of what data retrieved from DB files

⇒ Relational Calculus [uses FOL, predicate (first order logic) Calculus formula]

- Tuple Relational Calculus (TRC)
- Domain relational Calculus (DRC)
- SQL [IBM]
- QBE (Query By example) [microsoft]

Relational Algebra

Basic operators :-

Unary operator

- π : Projection
- σ : Selection
- $\not{p}$ X : Cross Product
- $\not{\rho}$: Rename
- $\cup$: Union
- $-$: Set difference

Derived operators

- $\cap$: Intersection (Using −)
- $\bowtie$: Join (Using X, σ, π)
- / or ÷ : Division (Using π, X, −)

Projection (π) :

$\pi_{list\_Attr}(R)$: Retrieve specific attributes from R

Selection (σ) :

$\sigma_P(R)$: Retrieve records of Rel R those are satisfied condition P.

R

A	B	C
4	2	4
5	2	4
6	2	5
6	2	5

$\pi_{BC}(R)$

B	C
2	4
2	5

} distinct tuples

$\sigma_{A \geq 5}(R)$

A	B	C
5	2	4
6	2	5

} distinct tuples

Q- Which is true

(i) π is commutative

(ii) σ is commutative

(a) (i) (b) ii (c) i, ii (d) none

$\phi_2\ \phi_1\ (R)$ → if order of opr^n does not matter then commutative

$\pi_{list_2}(\pi_{list_1}(R)) \neq \pi_{list_1}(\pi_{list_2}(R))$

$\pi_B(\pi_{BC}(R))$ $\quad$ $\pi_{BC}(\pi_B(R))$

Invalid Query

$\sigma_{L_1}(\sigma_{L_2}(R)) = \sigma_{L_2}(\sigma_{L_1}(R))$

$\sigma_{A \geq 6}(\sigma_{A \geq 5}(R)) = \sigma_{A \geq 5}(\sigma_{A \geq 6}(R))$

$\sigma_{center=kol}(\sigma_{Branch=cs}(R)) = \sigma_{Branch=cs}(\sigma_{center=kol}(R)) = \sigma_{C1 \wedge C2}(R)$

Q- Which of LHS Query can't replace with RHS Query $list_1$, $list_2$ attributes of R. C_1 C_2 conditions over Attr of R

(a) ✓ $\pi_{list_1}(\sigma_{C1}(R)) \longrightarrow \sigma_{C1}(\pi_{list1}(R))$

(b) $\sigma_{C2}(\pi_{list_2}(R)) \longrightarrow \pi(\sigma_{C2}(R))$ (with $list_2$)

(c) $\sigma_{C2}(\sigma_{C1}(R)) \longrightarrow \sigma_{C1}(\sigma_{C2}(R))$

(d) none.

R (Sid Sname age)

$\pi_{sid}(\sigma_{age \geq 20}(R)) \longrightarrow \sigma_{age \geq 20}(\pi_{sid}(R))$ (a) ✓ (b) ✗

$\sigma_{sid=S10}(\pi_{sid\,sname}(R)) \longrightarrow \pi_{sid\,sname}(\sigma_{sid=S10}(R))$ ✓

Cross – Product (×)

R × S : results all attributes of relation R followed by all attributes of relation S and each record of relation R pairs with every record of relation S

R

A	B	C
2	4	7
8	4	3
7	5	7

n distinct tuples

Arity : X

S

C	D
7	5
6	5

m distinct tuples

Arity : Y

R × S ⇒

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

Cardinality : n * m

Arity : x + y

eg:-

R

A	B
x	4
5	6

S

C	D
empty	

R × S

A	B	C	D

} empty

R × S product result empty record set if either R or S is empty record set

~~No~~ JOIN

- Natural Join ($\bowtie$)
- Conditional Join ($\bowtie_c$)
- Outer Join
  - Left Outer Join ($\bowtie$)
  - Right Outer Join ($\bowtie$)
  - Full Outer Join ($\bowtie$)

Natural Join :

$$R \bowtie S = \pi_{\text{distinct attribute}} \left(\sigma_{\text{equality b/w common attri of R \$S}} (R \times S) \right)$$

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

Rename operator (ρ):

Used for rename table / attributes for query processing.

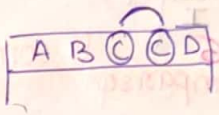
eg: stud (sid , sname , age)

ρ (temp , stud) : Temp (sid sname age)

ρ (stud) : stud (I , N, A)
I, N, A

ρ (stud) : stud (I, sname, A)
sid → I
age → A

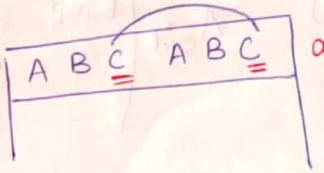
eg: R (A B C) S(C D)



$R \bowtie S$
$R.C > S.C$

eg:- R(A B C)

$R \times R$



ambiguity bec table is same

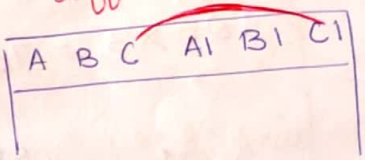
$\Downarrow$

$R \bowtie \rho(T,R)$
$R.C > T.C$

(or)

$R \bowtie \rho_{A1\,B1\,C1}(R)$
$C > C1$

⇒ now we can differentiate

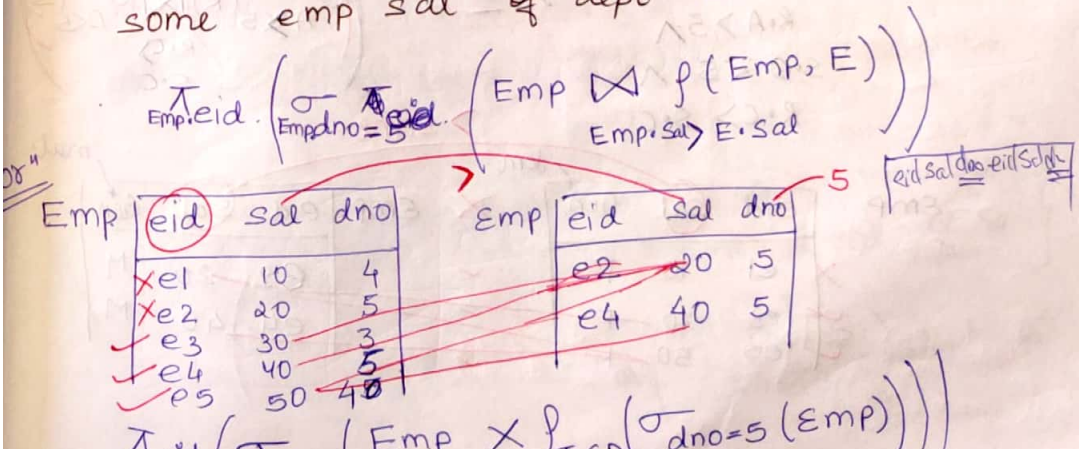


when we use two instance of same table, renaming is used.

eg:- Emp (eid sal dno)

Retrieve eid's whose salary more than some emp sal of dept 5

$\pi_{Emp.eid}\left(\sigma_{Emp.dno=5}\left(Emp \bowtie \rho(Emp, E)\right)\right)$
$Emp.sal > E.sal$



$\pi_{eid}\left(\sigma_{sal>S}\left(Emp \times \rho_{I,S,D}\left(\sigma_{dno=5}(Emp)\right)\right)\right)$

= some

Enroll (Sid Cid fee) Course (Cid Inst) Koath

Sid	Cid
S1	C1
S1	C2
S1	C4
S2	C2
S2	C3
S3	C4

Cid	Inst
C1	Koath
C2	Koath
C3	Navathe
C4	Ullman

Retrieve Sid's enrolled (some) course taught by Koath.

$$\pi_{Sid}\left((Enroll) \bowtie_{\substack{Enroll.Cid \\ = \\ course.Cid}} \left(\sigma_{Inst=Koath}(Course)\right)\right)$$

6*2=12 ⇒ 16 comparisons

Or

$$\pi_{Sid}\left(\sigma_{\substack{Enroll.Cid \\ = \\ course.Cid \\ \wedge \\ Ins=Koath}}(Enroll \times course)\right)$$

24 tuples in course product.

2 comp per record.

⇒ 48 comparison

some/ any/ atleast one.	$R \bowtie S$, $R \bowtie_c S$, $\sigma(R \times S)$

** Q- R(A....) S(B...)

Retrieve 'A' vals of R those are more than (every) 'B' of S.

≤ some

R(A) S(B...)

R(A): 10, 20, 30, 40

S(B...): 15, 25, 30

¬(> every) ≡ (≤ some)

{All 'A' val of R} − {'A' val of R which are ≤ some B val of S}

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

Q Emp(eid sal)

Retrieve eid's who gets highest salary (max sal)

(max^m salary) = (Sal ≥ every emp sal)

$$\text{Sal} \geq \text{every emp Sal} \equiv \{\text{all emp eid's}\} - \{\text{Sal} \leq \text{some emp sal}\}$$

// eid's whose salary is max^m.

// eid's whose salary is not max^m.

$$\pi_{eid}(Emp) - \pi_{eid}\left((Emp) \bowtie_{sal \leq S} \rho_{I,S}(Emp)\right)$$

NOTE:

R(A ...)

① [max 'A' val of R]

$$\{A \geq \text{every } A\} \equiv \{\text{All 'A' val}\} - \{\text{'A'} < \text{some } A\}$$

$$\pi_A(R) - \pi_A\left(R \bowtie_{A<A1} \rho_{A \to A1}(R)\right)$$

② [min 'A' val of R]

$$\{A \leq \text{every } A\} \equiv \{\text{All 'A' val}\} - \{\text{'A'} > \text{some } A\}$$

$$\pi_A(R) - \pi_A\left(R \bowtie_{A>A1} \rho_{A \to A1}(R)\right)$$

Q- Emp(eid sal dno)

Retrieve ~~eid~~ eid's whose sal is max for each dept.

$$\{\text{sal} \geq \text{every emp sal of dept same}\} \equiv \{\text{All Emp eid}\} - \{\text{sal} < \text{some sal of dept same}\}$$

// eid whose sal max for same dept

// eid who sal not max for same dept

$$\pi_{eid}(Emp) - \pi_{eid}\left(Emp \bowtie_{sal<S \,\wedge\, Dno=D} \rho_{I,S,D}(Emp)\right)$$

"or"

	emp(eid	sal	dno)		emp(eid	sal	dno)
e_1	e1	70	2			70	2
e_2	e2	80	2			80	2
	e3	65	2			65	2
e_3	e4	60	3			60	3
e_4	e5	70	3			70	3
e_5							

$= e_2 \not\in e_5$

domain of attribute
possible values accepted by attributes.

eg: $\pi_{sid,sname}(....) \cup \pi_{sid}(....)$ ✗ cond^1 failed

$\pi_{sid,sname}(...) \cup \pi_{sid,age}(...)$ ✗ cond^2 failed

$\pi_{sid,sname}(...) \cup \pi_{(studIDStudName)}(....)$ ✓ (Renamed) but their domains are same.

R ∪ S, R ∩ S, R − S set operations
(i) result distinct tuples
(ii) schema (table name, attribute names) of result same as schema of R.
Same in SQL

eg:- R

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

S

D	E	F
2	6	8
2	6	8
7	4	4
7	3	5

R ∪ S ≡ R [OR]

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

distinct tuples

R − S ≡ R [But not] [only]

A	B	C
7	2	4

R ∩ S ≡ R

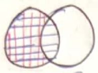
A	B	C
2	6	8
7	3	5

Q- R (A B)   S (C D)

A	B
2	4
2	5

C	D
2	4
3	6

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



$R \cap S \equiv R \bowtie \rho_{C \to A, D \to B}(S)$

$R \cap S \equiv \pi_{A,B}\left(R \bowtie_{A=C \wedge B=D} S\right)$

∴ "∩" is derived operⁿ.

Q - Enroll (sid cid)   Course (cid Inst)

Retrieve sid's enrolled some course taught by Koath or some course taught by Novathe.

(sid enrolled some Koath course) OR (sid enrolled some Novathe course)

$\pi_{sid}\left(Enroll \bowtie_{Enroll.cid = Course.cid} \left(\sigma_{Inst=Koath}(course)\right)\right) \cup$

$\pi_{sid}\left(Enroll \bowtie_{Enroll.cid = course.cid} \left(\sigma_{Inst=Navath}(course)\right)\right)$

"OR"

$\pi_{sid}\left(Enroll \bowtie \sigma_{Inst=Koath \vee Inst=Navathe}(course)\right)$

Q- Retrieve sid's enrolled some Koath course and some navathe course.

$\pi_{sid}\left(Enroll \bowtie \sigma_{Inst=Koath \wedge Inst=Navathe}(course)\right)$ ✗

"or"

$\pi_{sid}\left(Enroll \bowtie \sigma_{Inst=Koath}(course)\right) \cap \pi_{sid}\left(Enroll \bowtie \sigma_{Inst=Navathe}(course)\right)$ ✓

Q- Retrieve sid's enrolled only courses taught by Koath.

$\pi_{sid}\left(Enroll \bowtie \sigma_{Inst=Koath}(course)\right) \rightarrow \pi_{sid}\{\ \}$ — some Koath course ⋈ / (only) Koath course

{sid's enrolled only courses taught by Koath} = {sid enrolled some course} − {sid enrolled some non Koath course}

s1	s1
s2	s3
s3	

s1 c1 — c1 Koath
s1 c3 - c3 Navathe

only / but not : (−)

$\pi_{sid}(Enroll) - \pi_{sid}\left(Enroll \bowtie \sigma_{Ins \neq Koath}(course)\right)$

DIVISION (/ or ÷)

pair with every

Q. Enroll (Sid Cid) course (Cid ...)

Enroll	
S1	C1
S1	C2
S1	C3
S2	C1
S2	C2
S3	C1

course
C1
C2
C3

Retrieve sid's enrolled every course

$\pi_{sid,cid}(Enroll) / \pi_{cid}(course) \equiv$

Sid
S1

Expansion of / :-

⇒ Retrieve sid's enrolled every course

$\pi_{sid,cid}(E) / \pi_{cid}(C)$

not enrolled every S_2 S_3

≠

enrolled some S_1 S_2 S_3

(a) Retrieve sid's not enrolled every course [sid's enrolled proper subset of all courses]

$\pi_{sid}\Big(\pi_{sid}(E) \times \pi_{sid}(C) - \pi_{sid,cid}(E)\Big)$

S_2 S_3

$\pi_{sid}(E) \times \pi_{sid}(C)$: every sid of enrolled rel pairs with all courses

$\pi_{sid,cid}(E)$: Given enrolled records

disqualified sid's for division

(b) {Sid's enrolled every course} ≡ {sid enrolled some course} − {sid's not enrolled every course}

$\pi_{sid,cid}(E) / \pi_{cid}(C) \equiv \pi_{sid}(E) - \pi_{sid}\Big(\pi_{sid}(E) \times \pi_{sid}(C) - \pi_{sid,cid}(E)\Big)$

Q- Enroll (sid cid) course (Sid Inst)

Enroll	
S1	C1
S1	C2
S1	C3
S2	C2
S2	C4
S3	C4

course	
C1	Korth
C2	Korth
C3	Navathe
C4	Ullman

(a) Retrieve sid's enrolled some course taught by Korth.

(b) Retrieve sid's enrolled every course taught by Korth.

(a) $\pi_{sid}\Big(Enroll \bowtie \sigma_{Ins=Korth}(course)\Big)$

⇒ S1, S2

(b) $\pi_{sid,cid}(Enroll) / \pi_{cid}\Big(\sigma_{Ins=Korth}(course)\Big)$

⇒ S1 ≡

$\pi_{sid}(E) - \pi_{sid}\Big(\pi_{sid}(E) \times \pi_{cid}\big(\sigma_{Ins=Korth}(C)\big) - \pi_{sid,cid}(E)\Big)$

sid enrolled proper subset of Korth courses.

S1 S2 S3

S2 S3

S_1	C_1
S_1	C_2
S_2	C_1
S_2	C_2
S_3	C_1
S_3	C_2

S_2	C_1
S_3	C_1
S_3	C_2

Q- Stud.(sid gen)

Enroll (sid cid)

(a) Retrieve cid's enrolled some female stud

(b) Retrieve cid's enrolled every female stud.

(a) $\pi_{cid}\Big(Enroll \bowtie \sigma_{gen=F}(Stud)\Big)$

(b) $\pi_{cid}(E) \equiv \pi_{cid}(E) - \pi_{cid}\Big(\pi_{cid}(E) \times \pi_{sid}\big(\sigma_{Gen=F}(Stud)\big) - \pi_{E.cid,stud.sid}\big(\sigma_{Gen=F}(Stud)\big)\Big)$

$\pi_{sid}\big(\sigma_{Gen=F}(Stud)\big)$

cid's enrolled proper subset of all females

$\pi_{cid,sid}(E)$

Atleast two : $\sigma_p(T_1 \times T_2)$

Only two : Atleast two − Atleast three

Atmost two : All − Atleast three.

Q- Enroll (Sid Cid)

(a) Retrieve sid's enrolled at least 2 courses

NOTE:

FOL :-

fun : C(x) ⇒ x is student in class

S(x) x studied maths

⇒ some student in class studied maths

$\exists x\,(C(x) \wedge S(x))$

⇒ Atleast two students in class studied maths

Some x student in class studied maths (and) Some y student in class studied maths (and) $x \neq y$

$\exists x\,\exists y\,\Big((C(x) \wedge S(x)) \wedge (C(y) \wedge S(x)) \wedge (x \neq y)\Big)$

=

Enroll (Sid Cid) $\times_{\neq}$ Enroll (Sid Cid)

S1	C1
S1	C2
S1	C3
S2	C1
S2	C2
S3	C3

S1	C1
S1	C2
S1	C3
S2	C1
S2	C2
S3	C3

$\pi_{Sid.}\Big(\text{Enroll} \bowtie_{\substack{Sid=S \\ \wedge\ Cid \neq C}} \rho_{S,C}(\text{Enroll})\Big)$

(b) Retrieve sid's enrolled at least 3 courses

Sid	Cid	Sid	Cid	Sid	Cid
S1	C1	S1	C1	S1	C1
S1	C1	S1	C1	S1	C2
S1	C1	S1	C2	S1	C1
S1	C1	S1	C2	S1	C3
S1	C2	S1	C1	S1	C3

$T_1 \bowtie T_2$

$\pi_{Sid}\Big(\text{Enroll} \bowtie_{\substack{Sid=S1 \\ Cid \neq C1 \\ \wedge\ Cid \neq C2}} \rho_{S1}(\text{Enroll}) \bowtie_{\substack{Sid=S \\ Cid \neq C \\ C2 \neq S1}} \rho_{S,C}(\text{Enroll})\Big)$

"or"

$\pi_{T_1.Sid}\left(\sigma_{\substack{T_1.Sid = \\ T_2.Sid = \\ \wedge\ T_2.Sid = T_3.Sid}} \Big(\rho(T_1, \text{Enroll}) \times \rho(T_2, \text{Enroll}) \times \rho(T_3, \text{Enroll})\Big) \wedge\ T_1.Cid \neq T_2.Cid \wedge T_2.Cid \neq T_3.Cid \wedge T_1.Cid \neq T_3.Cid\right)$

Cost = 5 comparison $\times$ 216 records ⇒ 1080

NOTE:

⇒ Only one student in class studied maths

[atleast one student in class studied maths] but not [atleast two student in class studied maths]

$\exists x\,(P(x) \wedge \neg Q(x))$

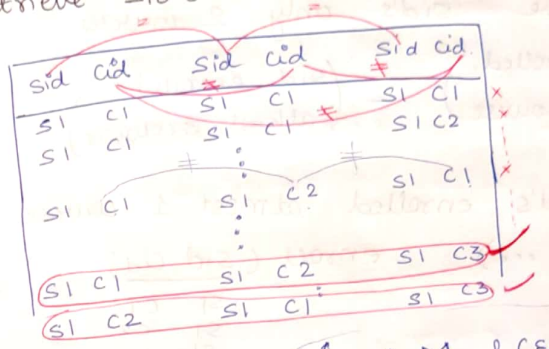
$\exists x\,(C(x) \wedge S(x)) \wedge \neg\,\exists x\,\exists y\,(C(x) \wedge S(x) \wedge C(y) \wedge S(y) \wedge (x \neq y))$

(c) Retrieve sid's enrolled only one course

(sid enrolled atleast one course) − (sid enrolled atleast two course)

$\pi_{Sid}(\text{Enroll}) - \pi_{Sid.}\Big(\text{Enroll} \bowtie_{\substack{Sid=S \\ Cid \neq C}} \rho_{S,C}(\text{Enroll})\Big)$

(b) Retrieve



$T_1 \bowtie T_2$

$$\pi_{sid}\left(Enroll \bowtie_{sid=S \wedge cid \neq C} \rho_{S1}\left(Enroll \bowtie_{sid=S \wedge cid \neq C} \rho_{S,C}(Enroll)\right)\right)$$

"or"

$$\pi_{T_1.sid}\left(\sigma_{\substack{T_1.sid = \\ T_2.sid \wedge \\ T_2.sid = T_3.sid}} \left(\rho(T_1, Enroll) \times \rho(T_2, Enroll) \times \rho(T_3, Enroll)\right)\right)$$

$T_1.cid \neq T_2.cid \wedge T_2.cid \neq T_3.cid \wedge T_1.cid \neq T_3.cid$

Cost = 5 comparison × 216 records ⇒ 1080

NOTE:

⇒ Only one student in class studied maths

[atleast one student in class studied maths] but not [atleast two student in class studied maths]

$\exists_x (C(x) \wedge \neg(\exists_y (C(y) \wedge (x \neq y))))$

$\exists_x (C(x) \wedge S(x)) \wedge \neg \exists_x \exists_y (C(x) \wedge S(x) \wedge C(y) \wedge S(y) \wedge (x \neq y))$

(c) Retrieve sid's enrolled only one course

(sid enrolled atleast one course) − (sid enrolled atleast two course)

$$\pi_{sid}(Enroll) - \pi_{sid}\left(Enroll \bowtie_{sid=S \wedge cid \neq C} \rho_{S,C}(Enroll)\right)$$

(d) Retrieve sid's only 2 courses

(sid's enrolled atleast 2 course) − (sid enrolled atleast 3 courses)

Q Retrieve sid's enrolled atmost 1 course

Stud (Sid ...)

S1
S2
S3
S4

enroll (Sid Cid)

S1	C1
S1	C2
S1	C3
S2	C1
S2	C2
S3	C3

(all students of Stud) − (sid's enrolled at least 2 courses)

$\pi_{Sid}(Stud) - \pi_{Sid}(Enroll \bowtie_{Sid=S \wedge Cid \neq C} \rho_{S,C}(Enroll))$

NOTE: Atmost (1)

$\exists x (C(x)) \wedge \neg \exists x \exists y (C(x) \wedge S(x) \wedge C(y) \wedge S(y) \wedge (x \neq y))$

Q- Stud (Sid Sname) with 200 tuples
Enroll (Sid Cid) with 100 tuples
How many (max, min) tuples in result of Stud ⋈ Enroll.?

Stud (Sid Sname) = Enroll (Sid Cid)

S1	A	S1	C1
S2	B	S1	C2
S3	A	S1	C3
S4	B	S2	C2
S5	C		
S6	C		
S7	A		

200 tuples

100 tuples

max = 100 tuples

min tuples = {100 FK exist; 0 no FK}

Q- R(A B C) with n tuples, no NULL values.
S(B D E) with m tuples

max tuples in R ⋈ S: n

min tuples in R ⋈ S: n → FK; 0 → no FK

R(A B C) = S(B A D E)

max = n.
min = m

Q R(A B C) with n tuples
S(A D E) with m tuples

max in R ⋈ S: min(m,n)
min in R ⋈ S: {min(m,n); 0 → (no FK)}

max = max(m,n)
min = min(m,n)

R(A ...) S(A ...)

Q R(A B C) with n tuples
S(D B E) with m tuples

max in R ⋈ S: mn
min in R ⋈ S: {→ 0}

A B C = D B E

Q R(A B C) with 1000 tuples
S(B D E) with 2000 tuples
functional dependencies {A → B, B → C, B → DE} exists.
How max tuples in R ⋈ S

1000

⇒ Rel R with X set of attributes && n distinct tuples

Rel S with Y set of attributes && m distinct tuples

(i) Required Condⁿ to perform R divide by S

~~X ⊇ Y~~

$X \supset Y$

(ii) Attribute set in result of R divide by S

$X - Y$

(iii) Cardinality (min, max tuples) of R / S

$\pi_{AB}(R) \;/\; \pi_{B}(S) \equiv$

A
empty

A	B
a1	2
a2	3
a1	3
a2	4
a3	2
a3	4

B
2
3
4

$\pi_{AB}(R) \;/\; \pi_{B}(S) \equiv$

A
a_1
a_2

$\lfloor \frac{n}{m} \rfloor$

A	B
a_1	2
a_1	3
a_1	4
a_2	2
a_2	3
a_2	4

$\{0 \text{ to } \lfloor n/m \rfloor\}$

Q- E (Sid, Cid)

Sid	Cid
S1	C1
S1	C2
S2	C2
S3	C3

C (Cid, Inst)

Cid	Inst
C1	Navathe
C2	Navathe
C3	Ullman

How many records in result RA query?

$\pi_{sid,cid}(E) \;/\; \pi_{cid}(\sigma_{Ins=Korth}(C))$

(a) 0 (b) 2 (c) 3 (d) 4

Relational Algebra Query equal SQL Query

SQL Query :-

SELECT DISTINCT A1, A2, ... An : Projection (π)

FROM R1, R2, ... Rm : CROSS PRODUCT (×)

WHERE (P); : Selection (σ) operator
each record.

RA Query :-

$$\pi_{A1 \ldots An} \left(\sigma_P (R1 \times R2 \times \ldots \times Rm) \right)$$

SQL : ____

*] SQL Query (RA logic)

2] SQL Query fun.

Q- Catalog (Sid, Pid) Parts (Pid, Color) — Pid = Pid, Color = red.

Catalog Sid	Catalog Pid
S1	P1
S1	P2
S1	P3
S2	P2
S3	P3

Parts Pid	Parts Color
P1	Red
P2	Red
P3	Green

Retrieve suppliers who supplied some red part

Sid
S1
S2

$$\pi_{sid} \left(Catalog \bowtie \sigma_{colour=Red} (Parts) \right)$$

"or"

$$\pi_{sid} \left(\sigma_{colour=Red} (Catalog \times Parts) \wedge Catalog.Pid = Parts.Pid \right)$$

```
SELECT DISTINCT SID
FROM Catalog C , Parts AS P
WHERE ( (C.Pid = P.Pid) and
        ( color = red) );
```

AS → used for renaming

S1	P1	P1 red
S1	P2	P2 red
S2	P2	P2 Red

SID: S1, S1, S2 ⇒ it is not distinct

SID: S1, S2 ⇒ SELECT DISTINCT

SELECT

Aggregate Func :-

COUNT ( )
SUM ( )
AVG ( )
MIN ( )
MAX ( )
⇒

Aggregate func^n compute aggregation of non-NULL values, i.e NULL values discard by aggregate func^n.

```
Create Table R
( A integer (3),
  B integer (3)
);
```

R

A	B	ROWID
20	5	1
40	10	2
NULL	5	3
NULL	NULL	4
80	5	5
40	20	6

Count (*) : # of records

Count (A) : # of non null val of "A".

Count (Distinct A) : # of distinct non null val of "A"

```
SELECT Count (*) A1,
       Count (A) A2,
       Count (DISTINCT A) A3
FROM R;
```

Select # from mandatory for each Select clause there must be a from clause

op :

A1	A2	A3
6	4	3

```
SELECT SUM(A) A1, SUM (DISTINCT A) A2,
       AVG (A) A3, AVG (DISTINCT A) A4,
       MIN (A) A5, MAX(A) A6
FROM R;
```

o/p:

A1	A2	A3	A4	A5	A6
180	140	45	46.6	20	80

$$AVG(A) = \frac{SUM(A)}{COUNT(A)} = \frac{180}{4} = 45$$

$$AVG(DISTINCT A) = \frac{SUM(DISTINCT A)}{COUNT(DISTINCT A)} = \frac{140}{3} =$$

```
Select Avg(A), B
From R;
```

⇒ (error) because result can't be put in tabular format

o/p:-

Avg(A)	B
45	5
	10
	5
	⋮

} NOT IN TABULAR FORMAT

```
Select A
from R
where A = max(A);
```

each record.

20 = max (20) ✓

40 = max (40) ✓

it will not throw any error

o/p:

A
20
40
80
40

∴ aggregate func^{n} usage in where clause is not useful.

② Allowed to use aggregate fun's in SELECT clause and aggregate fun computes aggregation of each group

③ Not allowed to select any other attribute in SELECT clause.

Q R

A	B	C
a_1	b_1	90
NULL	b_5	40
a_3	b_2	70
a_1	b_1	70
a_3	b_2	50
NULL	b_5	50
a_1	b_2	NULL

GROUP BY (A)

A	B	C
a_1	b_1	90
a_1	b_1	70
a_1	b_2	NULL
NULL	b_5	40
NULL	b_5	50
a_3	b_2	70
a_3	b_2	50

aggregate is separate

(a) o/p

A
a_1
NULL
a_3

(b) o/p

A	AVG(C)
a_1	80
NULL	45
a_3	60

(a) ③ SELECT A
① FROM R
② GROUP BY A;

*(b) SELECT A, AVG(C)
FROM R
GROUP BY A;

it is allowed only when group by exist

(c) ③ Select Avg(C)
① from R
② GROUP BY (A);

AVG(C)
80
45
60

A is not present in select clause.

(d) Select A
① from R
② GROUP BY (A,B)

B is not present in select clause

(e) Select B
from R
GROUP BY A;

selection do not contain A

(d) Select A, B
from R
GROUP BY A;

B cannot be present in select clause.

	SQL stand	SQL compiler
(c)	Invalid	AVG(C): 80, 45, 60
(d)	Invalid	A: a_1, a_1, NULL, a_3
(e)	Invalid	ERROR → for single group value of A, multiple values of B exist. eg:- $a_1 \rightarrow b_1, b_2$ ∴ result can't be in tabular format
(f)	Invalid	a_1, NULL, a_3 → can't be in tabular format

HAVING :-

Having clause used to select groups based on which are satisfied having clause cond^n.

Having clause cond^n is tested for each group.

Cond^n of having clause must be
1. over aggregation (or)
2. over func^n like some(), every() etc. (should not use direct attribute comp in having clause)

(a) Select A
from R
GROUP BY A
HAVING Avg(C)>50;

o/p

A
a_1
a_3

(b) Select A, AVG(C)
FROM R
GROUP BY A
HAVING some(C) > 55;

A	AVG(C)
a_1	80
a_3	60

(c) Select Avg(C)
from R
Group by A
having every (C) > 45

A
a_3

(d) Select A
from R
group by A
having (C > 55); error, it is not suitable for

Nested Queries

1. Nested Query without Co-relation (Inner Query independent of outer Query)

```
SELECT    Attributes
FROM      Tables
WHERE     Condition      (Inner Query)
GROUP BY  Attributes
HAVING    Condition
ORDER BY  Attributes ;
```

eg: Emp (eid sal)

eid	sal
e1	80
e2	90
e3	70
e4	90

```
Select eid
from   Emp
where  sal = (select max(sal)
              from emp);
```
② Bottom query, ① Top query

o/p

eid
e2
e4

```
Select eid
FROM   Emp, (select max(sal) Msal
             FROM emp) temp (a one table)
where emp.sal
      =
temp.Msal;
```

o/p

eid
e2
e4

emp × temp

eid	sal	Msal
e1	80	90
e2	90	
e3	70	
e4	90	

→ Execution Flow – Bottom to top
→ Inner query computes only once.

2. Co-related Nested Query: (Inner Query is dependent on outer Query)

→ Inner query where clause uses attribute defined in outer Query.
⇒ Co-related nested query allowed WHERE clause HAVING clause of outer Query

eg: Co-relation in where clause – it is co-related
R(A...) S(B...)

```
Select A
Top ① from R
Top ③ where ( select Count (*)
              from S
              where R.A < S.B ) < 3 ;
```
② Bottom

R	A
	20
	40
	60
	50
	10
	80

S	B
	20
	40
	60
	50
	10
	80

⇒ Execution Flow ⇒ Top – Bottom – Top for each record of outer query
⇒ If co-relation in where clause inner query recomputes for each record of outer query

⇒ If co-relation in having clause, Inner Query recomputes for each group of outer query

eg:- Co-relation in HAVING clause

Emp (eid dno sal gen)

it is not corelated as no relation b/w outer & inner query

```
Select dno
① from Emp T1
② Group By dno
③ having Avg(sal) > ( Select Avg(sal)
                      from Emp T2
                      where T2.gen = male
                      and T2.dno = T1.dno
                    );
```

Emp (T1)

dno	sal	gen
2	60	F
2	70	F
2	40	M
2	50	M
3	80	M
3	90	M
3	60	F
3	50	F

o/p

dno
2

↳ now it is corelated

⇒ Query retrieves dept no for which Avg(sal) of dept is more than avg(sal) of male emp of same dept

eg:- Select eid
from emp
where sal = (Select sal
from emp
where dno = 5);

ERROR

sal = 50 (can't compare)

Sal
20
40
60

Fun used for Nestid Query

- IN / NOT IN
- ANY
- ALL (*)
- EXISTS / NOT EXISTS

⇒ it is used when, we want to use comparision of outer query with set of values of inner query like previous example

best suitable for non-corelated nested Query

best suitable for corelated nested Query

eg: R(A) S(B)

```
Select *
① from R
where ( Select min(B)
① from S) < 50;
```

R(A)	S(B)
20	40
40	60
60	80
80	

R(A)	S(B)
× 20	60
× 40	80
× 60	
× 80	

⇑ it is syntactically correct
it will not throw any error
but result of Query is meaningless

IN FUN :

⇒ Used for membership test

⇒ X IN Y

Y: set of Y values of inner query

True if X value is member of Y set

if (X) value must be equal to some Y value.

⇒ IN fun can used for equi Join Queries

(Natural Join / Conditional Join with equal Condition)

eg:- Enroll (Sid, Cid) = some Course (Cid, Inst)

Retrieve Sid's enrolled some course taught by Korth

Select Sid
From Enroll
where Cid IN (Select Cid
or = ANY from Course
where Inst = Korth);

Enroll ⋈ σ(Course)
Ins=Korth.

eg:- R(A B) S(C D)

$\bowtie_c$: Some

(i) $\pi_{A,B}\left(\begin{matrix} R \bowtie S \\ R.A = S.C \\ \wedge \\ R.B = S.D \end{matrix}\right)$ (A,B) = some (C,D)

RNS

(ii) Select distinct A,B
RNS from R
where (A,B) IN (Select C,D
from S);

(iii) Select distinct A,B
from R
where (A,B) NOT IN (Select A,B
from R
where (A,B) NOT IN (Select C,D
from S));

RNS = R − (R − S)

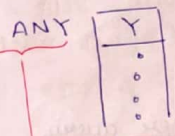
which is true

(a) Q_1, Q_2 results same but not Q_3
(b) Q_2, Q_3 same but not Q_1
(c) Q_1 Q_3 same but not Q_2
(d) Q_1 Q_2 Q_2 result same (d)

ANY && ALL fun:-

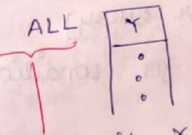
⇒ preceded by comparison operator <, <=, >, >=, =, <> (not equal.)

⇒ x > ANY Y



True if x value > atleast one Y value of Y set

x > ALL Y



True if x value > every value of Y set

⇒ ANY can used for conditional Join ($\bowtie_c$) Queries

eg: R(A ...) S(B ...)

(a) Retrieve 'A' val of R those are more than (some) 'B' val of S
$\bowtie_c$

$\pi_A\left(\begin{matrix} R \bowtie S \\ R.A > S.B \end{matrix}\right) \equiv$ Select A from R where (A) > ANY (select B from S);

(b) Retrieve 'A' val of R those are more than (every) 'B' of S

$\pi_A(R) - \pi_A\left(\begin{matrix} R \bowtie S \\ A \le B \end{matrix}\right) \equiv$ Select A from R where (A) > ALL (Every) (select B from S);

NOTE:
⇒ IN fun, (= ANY) func^n are same
⇒ (< > ALL) func^n can be replaced as NOT IN
⇒ ANY can be used for any form of cond^n join
⇒ ANY is used for only one attribute
⇒ IN can ~~be~~ have more than one comparison i-e, it can have set of attributes comparison

eg:- Enroll (Sid, Cid) = some Course (Cid, Inst)

Retrieve Sid's enrolled some course taught by Korth

Select Sid
From Enroll
where Cid IN / = ANY (Select Cid
from Course
where Inst = Korth);

Enroll ⋈ σ (Course)
Ins = Korth.

eg:- R(A B) S(C D)

$\bowtie_C$: Some

(i) $\pi_{A,B}\left(\begin{matrix} R \bowtie S \\ R.A = S.C \\ \wedge \\ R.B = S.D \end{matrix}\right)$ (A, B) = some (C, D)

R ∩ S

(ii) Select distinct A, B
R ∩ S from R
where (A, B) IN (Select C, D
from S);

(iii) Select distinct A, B
from R
where (A,B) NOT IN (Select A, B (R − S)
from R
where (A,B) NOT IN (Select C, D
from S));

R ∩ S = R − (R − S)

which is true

(a) Q_1, Q_2 results same but not Q_3
(b) Q_2, Q_3 same but not Q_1
(c) Q_1 Q_3 same but not Q_2
(d) Q_1 Q_2 Q_2 result same

(d)

EXISTS :

⇒ used to test result of inner query empty or not

⇒ EXISTS (Inner Query)

TRUE if result of inner query not empty

[Atleast some one record should in result of inner query]

⇒ EXISTS clause used for conditional Join Queries

Join Cond^n in where clause of inner query

eg: R(A...) S(B...)

10 ×, 20 ✓, 30 ✓, 40 ✓

15 ×, 25 ×, 30 ×

$\pi_A (R \underset{R.A>S.B}{\bowtie} S)$

(a) Select A
① from R
③ where exists (Select B
② from S
where R.A > S.B);

10 20 30 40

(b) Select A
from R
where NOT Exists (Select B
from S
where R.A ≤ S.B);

notexist, empty

R: A = 10, 20, 30, 40 (10 ×, 20 ✓, 30 ✓, 40 ✓)

S: B = 15, 25, 30

exist

A val. of R ≤ some B of

A val of R > every B of S

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

Q (a) select eid
from emp
when dno = 5;

$\pi_{eid}(\sigma_{d=5}(emp))$

e_1

> same

(b) Select *
from emp
where dno = 5;

e_1 5 80

$\sigma_{d=5}(Emp)$

Q ⇒ exist func^n for non-corelated query not meaningful.

Select eid
from emp T_1
where not exists (select *
from emp T_2
where T_2.dno = 5)

⇓
result is
empty or not empty
∴ meaningless

(c) Select distinct eid
from emp
where dno = 5;

(c) is more costly

From
⇓
where
⇓
Select
⇓
distinct
⇓

Acc. to Query Optimi, we should not use it until required.

Q- R

A	B
2	4
3	5
4	8

S

C	D
5	6
7	2
9	5

How many records in result of SQL Que

** $X = \{\ \}$ // empty

A op ANY X — False

A op ALL X — True

Sid

π_{sid} (Enroll ⋈ $\sigma_{ins=Korth}$ (Course)) — Some (∃)

empty

sid
S1
S2
S3

$\pi_{sid,cid} / \pi_{cid}$ ($\sigma_{ins=Korth}$ (Course)) — Every (∀)

Q- Emp(eid, sal)
Retrieve eid's who gets highest salary.

RA: π_{eid}(Emp) − π_{eid} (Emp ⋈ $_{sal<s}$ $\rho_{I,S}$(Emp))

SQL:
```
Select eid
from emp
except
Select T1.eid
from Emp T1, Emp T2
where T1.sal < T2.sal
```

SQL:
```
Select eid
from emp
where sal = ( Select max(sal) from emp);
```

SQL:
```
Select sid
from emp
where sal ≥ ALL ( Select sal from emp);
```

Q- Emp(eid sal)
Retrieve eid's who gets 2nd highest sal

ρ(Temp, $\pi_{eid, sal}$ (Emp ⋈ $_{sal<s}$ $\rho_{I,S}$(Emp))) → it is computed

emp

e1	60
e2	90
e3	70
e4	50

Temp

e1	60
e3	70
e4	50

π_{eid}(Temp) − π_{eid} (Temp ⋈ $_{sal<s}$ $\rho_{I,S}$(Temp)) ← 3 times computation needed.

to solve this problem.

With clause:
Sub-Query result of with clause can reuse many times for query processing.
To avoid re computation of same sub query with clause is used.

```
WITH temp (eid, sal) as
    ( select distinct T1.eid, T1.sal
      from emp T1, Emp T2
      where T1.sal < T2.sal)
Select eid
from temp
[where sal = (select max(sal) from temp);]
    or
except
select T1.eid
from Temp T1, Temp T2
where T1.sal < T2.sal ;
```

Using Nested Query with aggregation

```
Select eid
from emp
where sal = ( Select max(sal)          -- 70
              from Emp
              where sal < ( Select max(sal)   -- 90
                            from Emp ) );
```

~~e1~~	~~60~~
~~e2~~	~~90~~
✓e3	70
~~e4~~	~~50~~

✓e1	60
~~e2~~	~~90~~
✓e3	70
✓e4	50

K^{th} max

Nested Query
- (K+1) level
- (K+1) instances of emp

Co-related
- 2 level
- 2 instances of emp

Co-related Query

for 2nd max. (K^{th} max)

of T_2 distinct sal such that $T_1.sal < T_2.sal$

Emp (T_1) eid	sal
e1	70
e2	70
e3	60
e4	60
e5	50
e6	40
e7	30

Emp (T_2) eid	sal
e1	70
e2	70
e3	60
e4	60
e5	50
e6	40
e7	30

} distinct

```
Select eid
① from emp T1
③ where ( select count(distinct T2.sal)
  ② from emp T2
    where T1.sal < T2.sal ) = 0          -- (K-1)
```

high

K^{th} high

Q- Emp(eid dno sal)
Retrieve eid's who gets max^m salary for each dept

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

Using Group By and Aggregation

emp (eid	dno	sal)		Temp (dno	Max sal)
✗e1	2	70	=	2	90
✓e2	2	90		3	60
✗e3	2	60			
✗e4	3	40			
✓e5	3	60			

```
Select eid
from emp, ( Select dno, max(sal) MaxSal
            from Emp
            group by dno) temp
Where Emp.dno = temp.dno
  and sal = MaxSal ;
```

Q family (parent, child, child DOB)
Retrieve youngest child of each parent

```
select child
from family, ( select parent, max(DOB) DOB
               from family
               group by parent ) temp
Where family.parent = temp.parent
  and child DOB = DOB ;
```

"or"

```
Select child
from family
where (parent, child DOB) IN ( Select parent, max(child DOB)
                               from family
                               Group by parent );
```

Q- Works (eid pid)

Retrieve eid's who works for only one project

```
Select eid          <-- eid's works for atleast one project
from works
except
Select T1.eid       <-- eid's works for atleast 2 project
from Works T1, Work T2
where T1.pid <> T2.pid
and T1.eid = T2.eid
```

Using Group by and Having Clause.

Work (eid pid.)

eid	pid
e1	p1
e1	p2
e2	p2
e3	p2
e3	p3
e3	p4

having count(*) = 1

```
Select eid
from work
GROUP by eid
having count(*) = 1;
```

over aggregation

Q ~~em~~ same question

just add same dept. in the last

```
Select dno
from emp T1
where gen=F
group by dno
having (Avg(sal) >( Select Avg(sal)
                    from emp T2
                    where gen=M
                    and T2.dno = T1.dno);
```

"or"

```
Select dno
from ( select dno, Avg(sal) Avg Sal
       from emp
       where gen=fem
       group by dno) temp

where Avg Sal > (Select Avg(sal)
                 from emp
                 where gen=male
                 and emp.dno = temp.dno);
```

SQL Queries equal to Division (/) of RA:

Enroll (sid cid) Course (cid Instr)

Retrieve Sid's enrolled every course taught by Korth:

1] SQL co-relatid Query

Q $\Pi_{AB}(R) \div \Pi_{B}(S)$

```
Select distinct A
from R T1
where not exists ( select B
                   from S
                   except
                   select B
                   from R T2
                   where T1.A = T2.A
                 );
```

UNION
INTERSECT
EXCEPT

ER Model

Entity Relationships Diagrams used for high level DB design Diagrametic Representation of DB design

DB design steps:

1. Requirements → what type of data stored in DB
   → Type of operations etc

(High level DB design)

2. Conceptual & logical DB design : Entity sets && Relationships sets (ER diagram)
   → RDBMS tables

(Low level DB design)

3. Normalization : To eliminate/reduce redundancy
4. Physical DB design : Design of index.

5. Security && Application Design

(Low level DB design)

Main Components of DB Design:

1. Attributes
2. Entity Set
3. Relationship Sets

Attributes :-

Attribute

Key attribute

Multi Valued Attribute

Derived Attribute : Value of attribute derived from other stored Attribute

eg:- DOB → (age) age = sysdate - DOB

Composite Attribute : Attribute which can represent as two or more attribute

eg:-

Entity Set : Set of similar entities (tuples) represent by

eg:-

Relationship set :

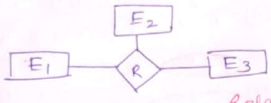
Used to relate two or more entity set

Represented as ◇

eg:- 

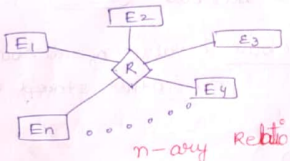
⇒ 2 foreign key

Binary Relationship Set.

eg:- 

⇒ 3 f_key

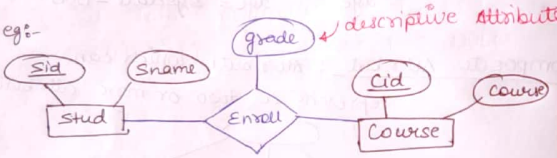
Ternary Relationship Set

eg:- 

⇒ n f_key

n-ary Relationship Set

E

eg:- 

Enroll (Sid Cid grade)

Integrity constraints for Relationship Sets

- Foreign key

Participation :

If every entity of entity set must related to relationship set then total participation otherwise partial participation.

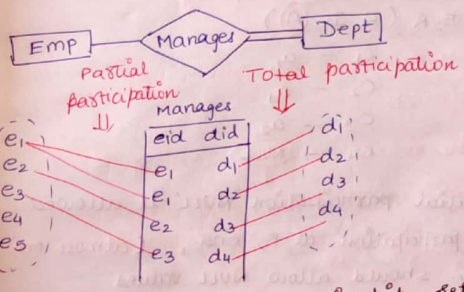
(═══) Total participation: must be 100% participation

(——) Partial participation: may / may not 100% participation

Q- Requirement :

Emp & dept are entity sets manages relationship set used to relate emp and dept such that each dept there must be one manager.

Ans:-



Cardinality (mapping) of Relationship set :

Mapping
- One (almost one)
- many (0 or more)

Mapping for Binary Relationship Set :-

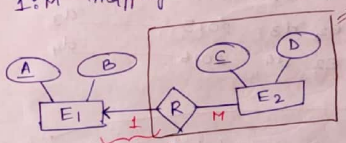
1. One : One mapping
2. One : Many mapping
3. Many : One mapping
4. Many : Many mapping

Candidate Keys of relationship set is based on Mapping

One : Many Mapping :-

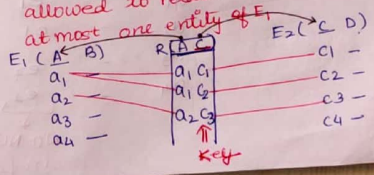
eg:- E_1 E_2 entity sets and R Rel set related b/w E_1 E_2 with 1:M mapping.

⇒ Min 2 RDBMS tables required and 1 foreign key



each entity of E_1 can relate many entity of E_2

E_2 entity allowed to relate at most one entity of E_1



For one to many mapping. candidate key of relationship set is key of many side (right side entity set

FD for E_1 : $\{A \rightarrow B\}$

E_2 : $\{C \rightarrow D\}$

R : $\{C \rightarrow A\}$

RDBMS design

foreign key.

E_1(A B) E_2R (C D A)

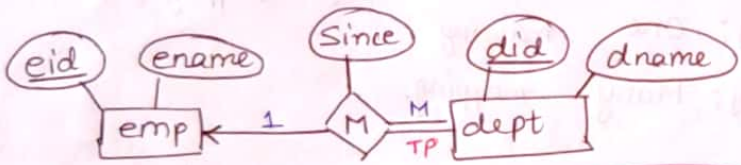
a_1	-	c_1	a_1
a_2	-	c_2	a_1
a_3	-	c_3	a_2
a_4	-	c_4	NULL

because of partial participation "NULL" is allowed

i.e, if Partial participation at E_2 end, (atmost one)
then F_key should allow NULL values

and, if Total participation at E_2 end. (exactly one)
then, F_key must be NOT NULL.

Q- Given ERD



Emp(eid ename) M(eid did since) dept(did dname)

Emp		M			dept	
e_1		e_1	d_1	2010	d_1	
e_2		e_1	d_2	2015	d_2	
e_3		e_2	d_3	2015	d_3	
e_4		e_2	d_4	2016	d_4	

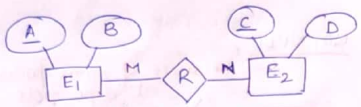
RDBMS design

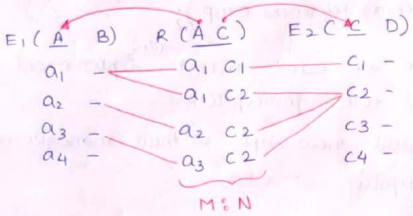
Emp(eid ename) Dept_M(did dname eid since)

d_1	-	e_1	2010
d_2	-	e_1	2015
d_3	-	e_2	2015
d_4	-	e_2	2016

NO NULL value

Many : Many Mapping :-





For many to many mapping C_Key of relationship set AC

FD for E_1 : $A \rightarrow B$

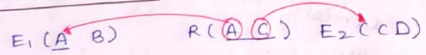
R : No non trival FD

E_2 : $C \rightarrow D$

Drawback:
if R combines with E_1 :- partial participation at E_1 voilate
RE_1 (A B C)
C_K : AC
$A \rightarrow B$
Data redundancy due to PD

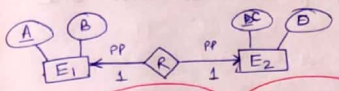
if R combines with E_2 : same drawback
RE_2 (A C D)
$C \rightarrow D$

RDBMS design

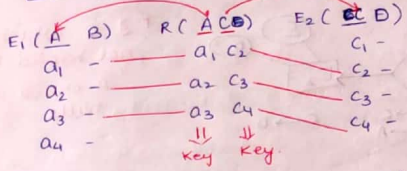


min 3 RDBMS tables and 2 F_Key's.

1 : 1 Mapping :-



1:1 Mapping
PP at both ends
- min 2 Rel tables and 1 foreign key.
- 2 cand_key req. required.



For 1:1 mapping C_keys of relationship set R is {A, C}

FD for E_1 : $A \rightarrow B$

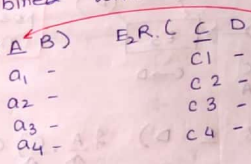
E_2 : $C \rightarrow D$

R : $A \rightarrow C$, $C \rightarrow A$

R can combine either with E_1 or with E_2

if Combined with E_2 :

RDBMS design:

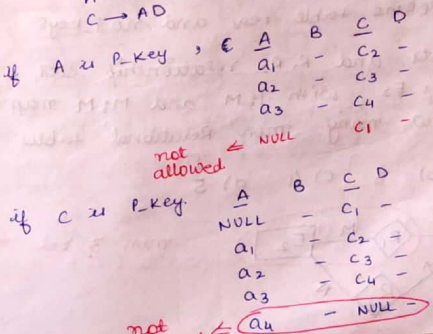


{A → C, C → AD}

A, C ⇒ C_key
Alt_key
P_key

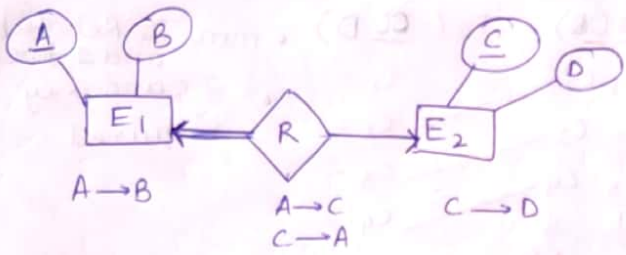
In 1:M → t_Key is not unique
1:1 → t_Key is unique

Drawback:
if E_1, E_2, R combines in one table :-
$A \rightarrow BC$
$C \rightarrow AD$
C_K = {A, C}



E_1, E_2 entity sets, relationship set R rela

E_1 & E_2 with 1:1 mapping and at least end total participation.



$E_1 R E_2$ (A B C D) { A, C } C_keys

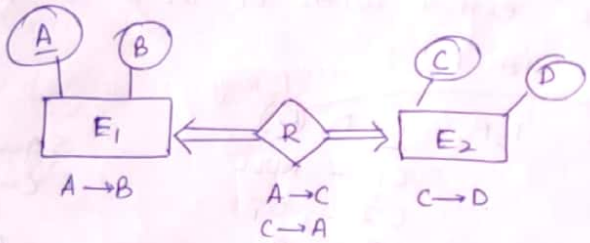
A	B	C	D
NULL	NULL	c_1	-
a_1	-	c_2	-
a_2	-	c_3	-
a_3	-	c_4	-

bec. of PP at E_2 side

A → Alt_Key, C → P_Key.

$E_1 : n$
$E_2 : m$
$(n \leq m)$

min 1 table req and no f_key.



possible or $(n = m)$

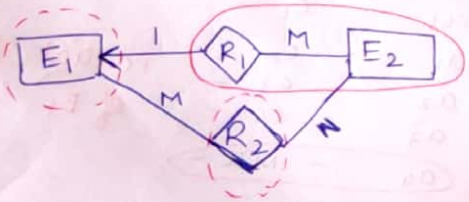
$E_1 R E_2$ (A B C D) { A, C } C_keys

A: Key, No NULL (PK)
C: Key, No NULL (UNIQUE NOT NULL)
(OR)
A: (UNIQUE NOTNULL)
C: (PK)

min 1 RDBMS table req and no f_keys.

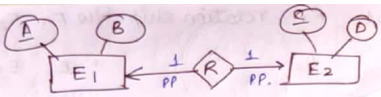
Q- E_1 E_2 entity sets and R_1 R_2 relationship sets related b/w E_1, E_2 with 1:M and M:M mapping respectively. How many min Relational tables req.

a) 2 b) 3 c) 4 d) 5



min 3 rel table

Q-



30% partici at E_1 end = PP

60% partici at E_2 end = PP

a) E_1, E_2 sepa. with F_K in E_1

b) E_2 E_2 sep with F_K in E_2

× c) E_1 E_2 merges in 1 table with no F_K
↓↓ only when TP at one end.

d) none

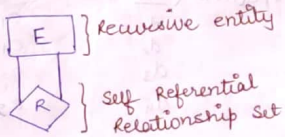
(a) R E_1 (A B C) E_2 (C D)

(b) R E_2 (C D A) E_1 (A B)

Self Referential Relationship set :-

Entities of entity set E related to some other entities of same entity set E

Can be
1:1
1:M
M:1
M:N mapping

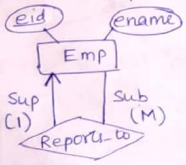


Q- given specification

Emp(eid, ename) entity set.

Reports_to relationship set related b/w supervisor and subordinate.

i) each supervisor can supervise many sub-ordinates and each sub-ordinate reports_to one supervisor.



Emp(eid ename)

e_1	A
e_2	A
e_3	B
e_4	C

Reports_to (Sup Sub)

e_1	e_2
e_1	e_3
e_2	e_4

Emp_Reports

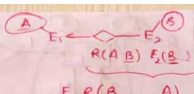
eid	ename	Sup
e_1	A	NULL
e_2	A	e_1
e_3	B	e_1
e_4	C	e_2

↑K

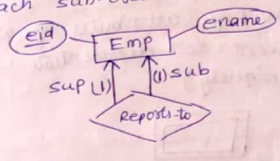
min 1 RDBMS table && 1 f_key

↑ eid of emp
sub of Reports_to

↓↓ not unique



ii) each supervisor can supervise one sub-ordinate and each sub-ordinate reports_to one supervisor



Emp (eid ename)

e_1	A
e_2	A
e_3	B
e_4	C

Reports_to (Sup Sub)

e_1	e_2
e_2	e_3
e_3	e_4

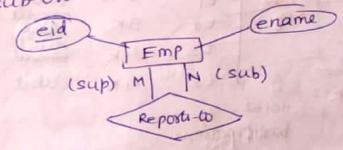
Emp_Reports ↑K

eid	ename	Sup
e_1	A	NULL
e_2	A	e_1
e_3	B	e_2
e_4	C	e_3

↓↓ unique

Cand_key = { eid, sup }

(iii) each supervisor can supervise many sub-ordinate and each sub-ordinate reports_to many supervisor



Emp(eid ename)

e_1	A
e_2	A
e_3	B
e_4	C

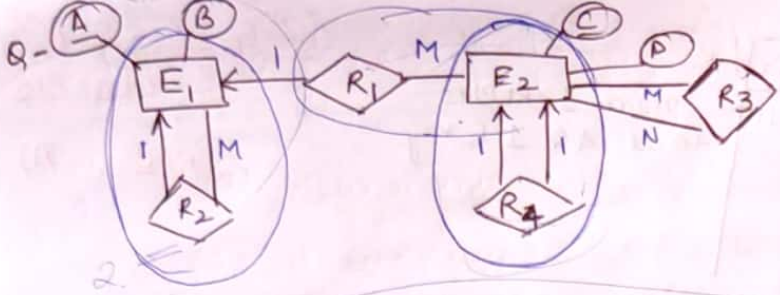
Reports_to (Sup Sub)

e_1	e_2
e_1	e_3
e_4	e_3

Emp_Reports
↓↓ we can't combine.

min 2 relational tables and 2 foreign keys

Cand_key { Subsup }



5 tk 9 attri
3 tables

$E_1 R_2$ (A B O) $R_1 E_2 R_4$ (C D O A) R_3 (O O)

A.K.

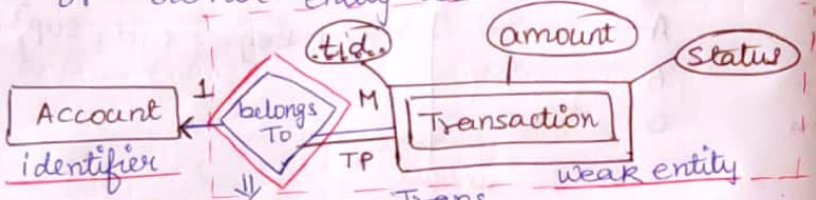
Weak Entity set:

Entity set with no key. (attributes of weak entity set not sufficient to differentiate entities of weak entity set uniquely)

Representation:

entities of weak entity set depends on other strong entity set entities which is called Identifier or owner entity set.

eg:-



aid .....
a_1
a_2
a_3
a_4
a_5

tid	aid
t_1	a_1
t_1	a_2
t_2	a_2
t_2	a_3

partial key

Weak

tid	amount	status
t_1	5 K	debit
t_1	5K	credit
t_2	4k	debit
t_2	4K	debit

total participation

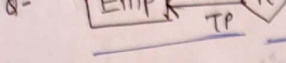
RDBMS design

account Trans belongsto

aid ....

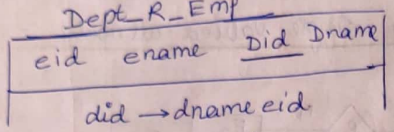
tid	aid	amount	status

min 2 rel. tables.



ⓐ How many min rel table

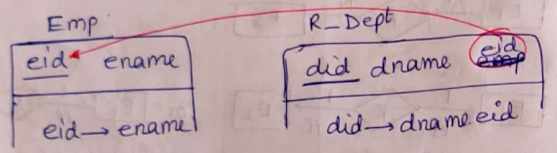
FD of Emp: eid → ename
Dept: did → dname
R: did → eid



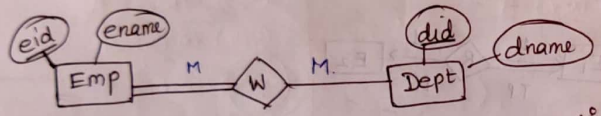
min 1 table
INF ✓
2NF ✓
3NF ✗

ⓑ How many min rel required which satisfy ~~BCNF~~ 3NF?

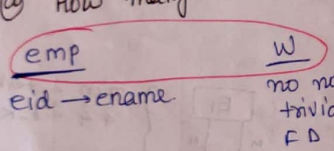
min 2 table



Q-



ⓐ How many min rel tables which satisfy INF?

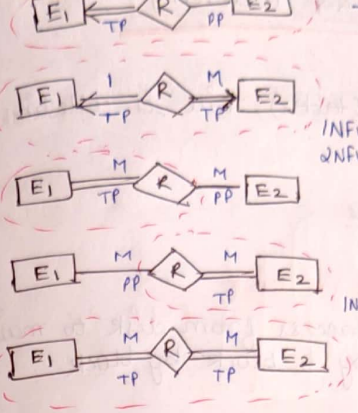


• min 2 tables req. INF ✓ 2NF ✗

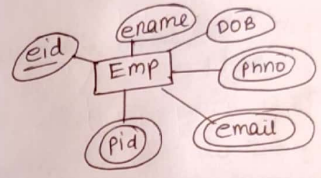
Emp_W (eid ename did) Dept (did dname)
eid → ename did → dname

Cand key: Eid did

ⓑ How many min rel tables which satisfy 2NF?



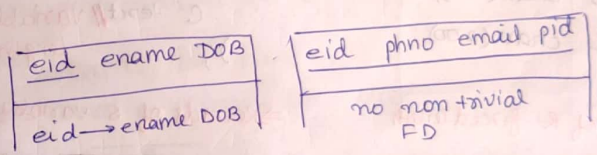
Q-



min rel tables req for INF?
1

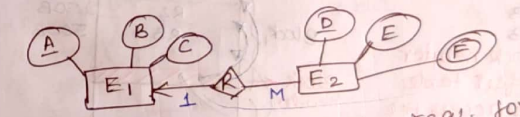
{eid → ename DOB}
FD (single valued Dep)
Emp (eidpno email pid ename DOB)
eid → ename DOB

INF ✓
2NF ✗
⇓
2 rel table



⇒ 2NF ✓ 3NF ✓ BCNF ✓

Q-



How many rel. tables req. for 3NF

E1: A → BC
R: D → A
E2: D → E

D → F : we need not have to bother for INF 2NF 3NF BCNF

1:M mapping combine with RHS entity set

min 2 table INFV
E1(A BC)
E2 R(D E F A)
D → E A
Cand key: {DF}

E1(A B C)
A → BC

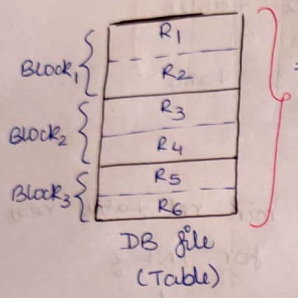
D E A
D → AE

D E
no non-trivial FD's

∴ – BCNF ✓
min 3 table for 3NF as well as BCNF

File Organisation && Indexing:

DB is collection of files (tables). File is collection of blocks (pages).
Block is set of records



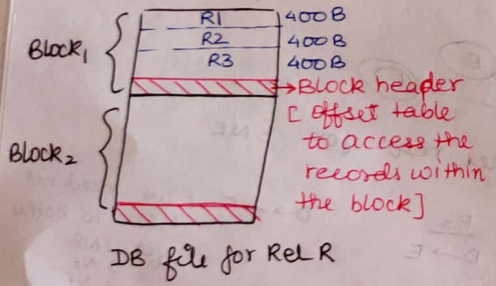
stored in Disk
⇒ data access from disk to main memory, block by block

Records in DB file can be

① Fixed length record

```
Create table R
(
    A  char(100),
    B  char(100),
    C  char(200)
);
```

⇒ Records of R fixed length

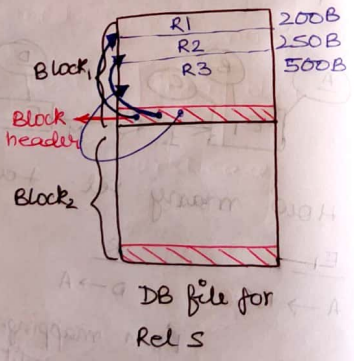


⇒ Unspanned org is preffered.

② Variable length record

```
Create table S
(
    A  char(100),
    B  char (100),
    C  text // variable length
);
```

⇒ Records of S variable length



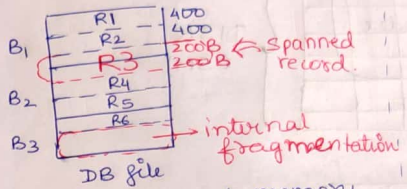
⇒ spanned org is preffered.

Records Organization in DB table

① Spanned org

Record allowed to span in more than one block

eg: Block : 1000 B
Record : 400 B



Adv: • no wasted of memory
• no memory hole
• disk utilisation is good.

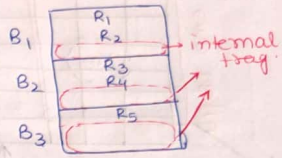
Possible to allocate DB file without internal fragmentation except for last block of the file

Disadv:
• Access cost is more to access the spanned records because we need to access 2 blocks in order to get R3

② Unspanned org.

Complete record must be stored in one block. i.e, record not allowed to span.

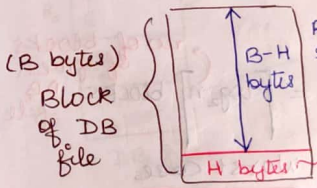
eg: Block : 1000 B
record : 400B



Disadv: more internal frag. which leads to wastage of disk space.

Adv: Access cost is less.

Block Factor of DB file :-
max possible records per block



Block factor of DB file $= \left\lfloor \frac{B-H}{R} \right\rfloor$ records/block

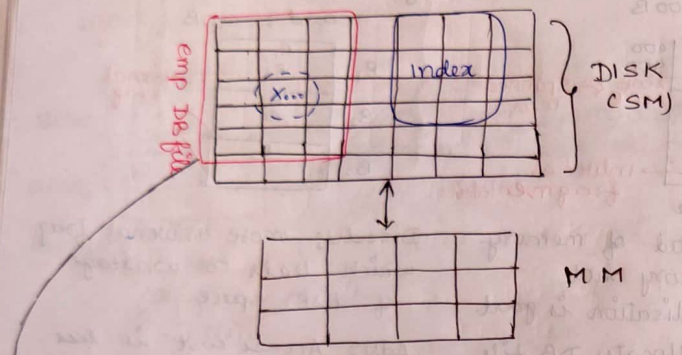
// because it is unspanned org

eg:- $B_f = \left\lfloor \frac{B-H}{R} \right\rfloor = \left\lfloor \frac{1000-0}{400} \right\rfloor = 2\ R/Block$ ⇐ fixed ~~variable~~ length or Unspanned org.

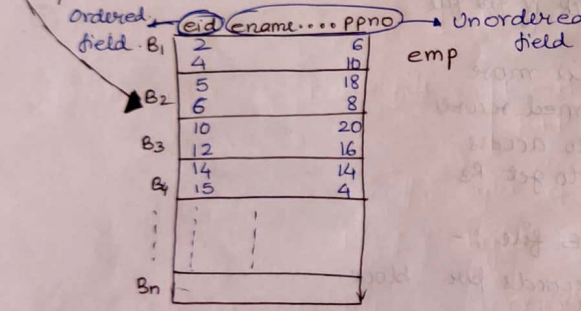
$B_f = \frac{B-H}{R} = \frac{1000-0}{400} = 2.5\ R/Block$ ⇐ spanned org.

Access Cost (I/O Cost):

no. of secondary memory blocks (disk blocks) required to transfer from disk to main memory in order to access the required records.



I/O cost to access record without index :-



a) based on ordered field.

```
Select *
from emp
where eid = x;
```

ordered field.

I/O Cost = $\lceil \log_2 n \rceil$ blocks

n = no of blocks of given DB file

Worst Case

b) based on Unordered field.

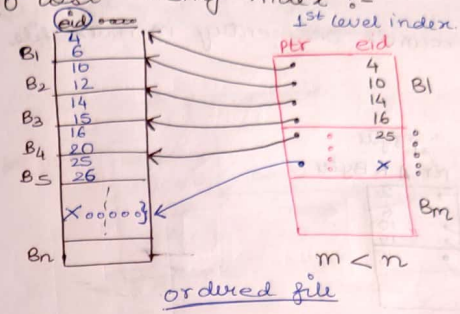
```
Select *
from emp
where ppno = y;
```

unordered field

I/O Cost = N blocks

Worst Case

I/O cost using Index :-



ordered file

I/O Cost to access record using 1st level indexing

: $\lceil \log_2 m \rceil + 1$ blocks

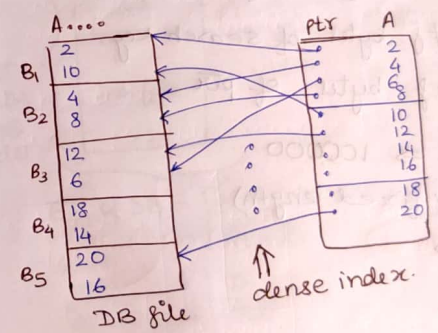
⇒ N
Q
ERD } develop

* Index
* T&C } build components DBMS SW

Categories :-

Dense Index:

Index entry for each record of DB file



# of index entities in index file = # of records of DB file

Dense index can build over ordered field and over unordered field.

Sparse Index :
For each of DB records one entry in index file

# of index file = # of blocks of DB file

Block factor of DB file $= \left\lfloor \frac{B-H}{R} \right\rfloor$ records/block

$\Rightarrow$ Block factor of index file $= \left\lfloor \frac{B-H}{K+P} \right\rfloor$ Entries/block.

K : # of bytes of search key

P : # of bytes of ptr

B- # of records in DB file $\Rightarrow$ 100000

Record size : 100 B (fixed length)

Block size : 1024 B

Search key : 12 B

Pointer : 9 B

① If dense index used
find. # of block of 1st level index

② If sparse index used
find # of blocks of 1st level index.

(a) # of records $= \frac{1024}{100} \Rightarrow$

Block factor $=$

(b)

$B_f = \left\lfloor \frac{B-H}{K+P} \right\rfloor = \left\lfloor \frac{1024-0}{12+9} \right\rfloor = 48$ entries/block

of ~~sparse~~ dense index blocks $= \left\lceil \frac{100000}{48} \right\rceil$

$= 2084$ blocks

(b) # of sparse blocks $\Rightarrow$

B_f of DB $= \left\lfloor \frac{B-H}{R} \right\rfloor$

$= \left\lfloor \frac{1024}{100} \right\rfloor$

$= 10$ R/block

no. of blocks $\Rightarrow \frac{100000}{10}$

$= 10,000$ blocks

# of sparse index blocks $= \left\lceil \frac{10,000}{48} \right\rceil = 209$ blocks

Multilevel Index :
Index to Index until last level one block of Index

I/o cost of to access records using multilevel Index : (K+1) blocks

⇒ to access 8 we need to access 3 blocks

Types of Index :

Search key : Ordered field — Search key : Un ordered field

Search key :
① Primary key/Unique key
[KEY]
↓↓
① Primary Index

Search key
[Non key]
↓↓
② Clustering Index

Search key :
PK / AK
[KEY]
↓↓
③ Secondary Index
[Duplicate key]

Search key
[Non key]
↓↓
④ Secondary Index
[Duplicate Not key]

Primary key Index

Primary Index can be either dense or sparse (sparse PI index is preffered) Atmost one PI is possible for any relation (DB file)
because it will reduce block number and level in MLI
search key must be ordered field.

I/o cost to access records using PI
using MLI (K+1) blocks

Clustering Index

eg: I/o cost
select * from emp dno = 3;

I/O cost to access one cluster of records using clustering with MLI =

K + one or more block access of DB, until first record of next cluster

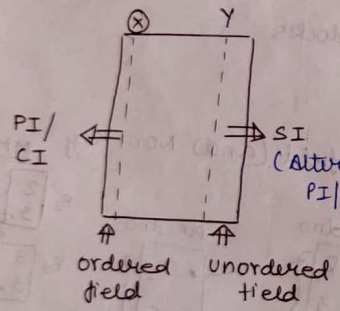
Clustering Index mostly sparse Index but dense clustering index may occurs if each cluster with 1 record

Atmost 1 clustering index is possible for any DB file because search key must be ordered field.

For any DB file either Primary Index / Clustering index is possible but not both.

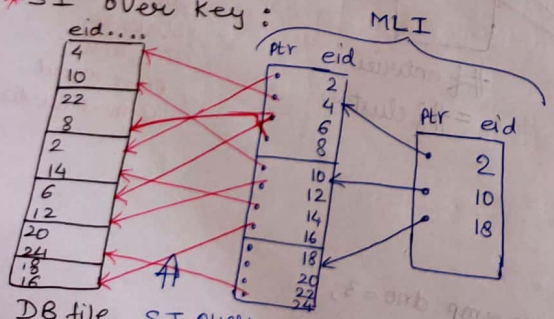
Secondary Index :

Search key : Unordered field (and) key / Non-key



⇒ Many SI possible for same DB file

*SI over Key :

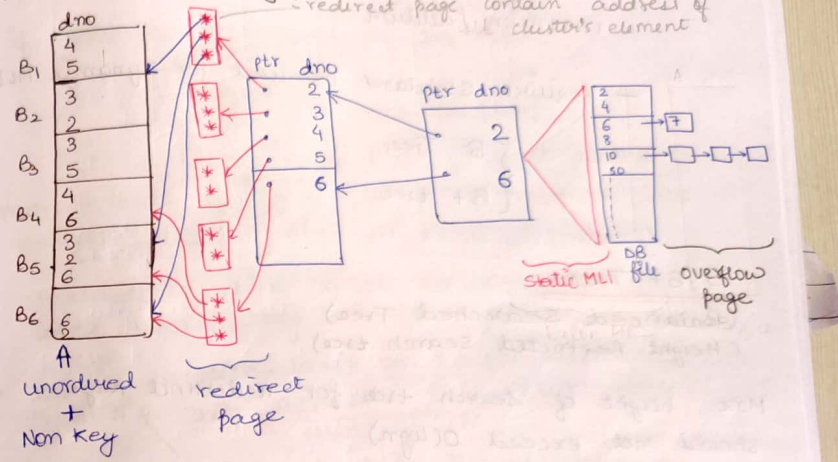


it is not suitable for frequent insertion and deletion

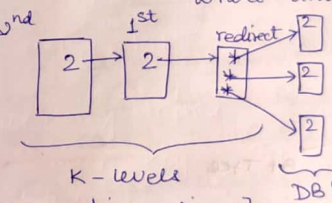
Secondary Index over key is always dense index.

I/O cost to access record using SI with MLI = (K+1) blocks

SI over Non-Key :



I/O Cost : Select * from emp where dno = 2;



⇒ { K + #f blocks of DB is #f records of given non key } blocks

Static MLI :

I/O cost can be O(n) ← Disadv.
because of overflow pages

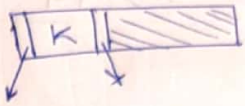
To avoid this disadvantage we go for dynamic MLI

If we add another entry in SI over key we need to change index entries and hence all MLI will change. so to avoid this we maintain overflow page so that we need not change index & we keep new entries in overflow page. So there may be many overflow page in same block. ~~Hence use office~~

Why B/B+ tree used for DB Index rather than Balanced BST ?

- In AVL, one block contain 1 ~~data~~ key
- In B+, one block contain multiple keys

- if Balanced BST used for DB index:

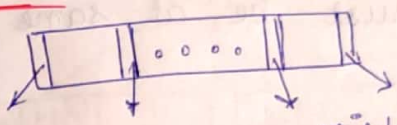


Block [AVL tree Node]

∵ it contain only 1 key, so it contains more levels of index which increases I/O cost to access data

- Wastage of disk space bec. each index block used for 1 key. remaining spaced is unused.
- DB file in disk. Index to DB file also must be in Disk && Data accessed from disk block by block

if B/B+ Tree index used for DB ~~fi~~ index.



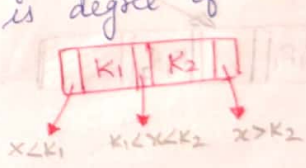
Block [B Tree node/ B+ Tree node]

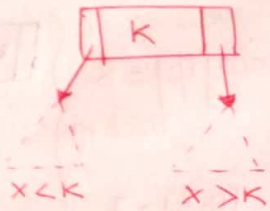
∵ it contain multiple keys in one block. so less levels of index hence reduces I/O cost to access data.

Also less wastage of disk space because each index block can store many keys & not one index block for one key.

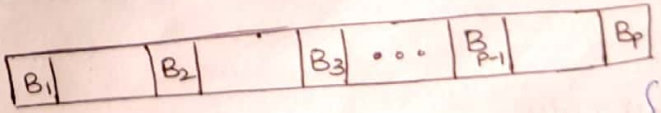
B Tree Def:

Order P: max possible child pointers (block pointer) which can store in B tree node is degree of B tree.



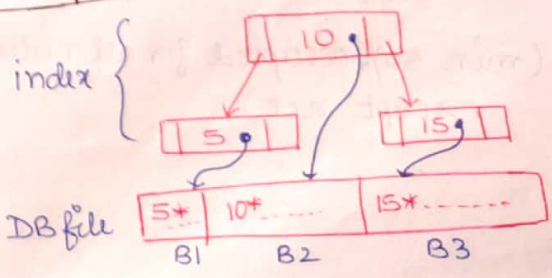


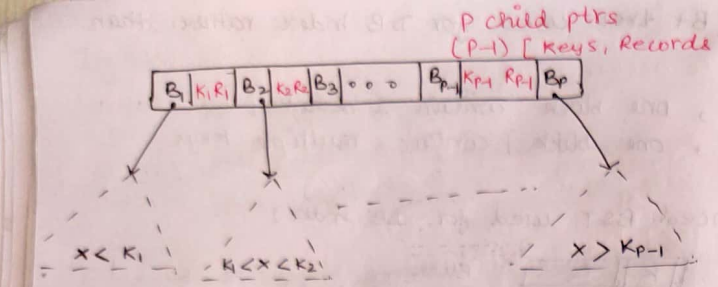
a) node structure:



Pointers :-

- pointer to child node
- points to DB file





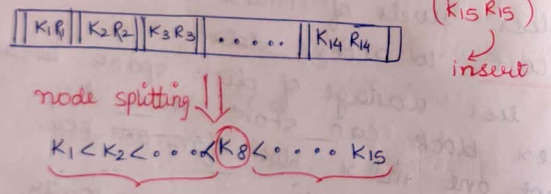
and b) Balancing condition:

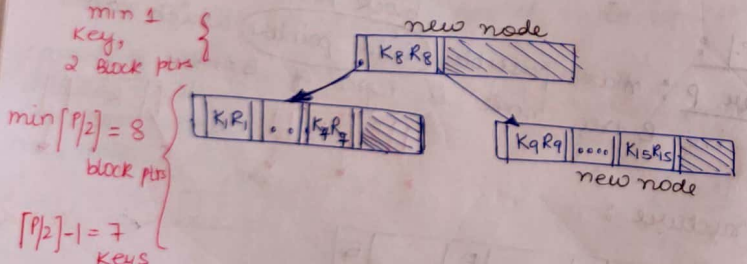
- every internal node except Root
  atleast $\lceil p/2 \rceil$ blocks ptrs with $(\lceil p/2 \rceil - 1)$ keys
  atmost p block ptrs with (p-1) keys

- Root can be atleast 2 block ptrs with 1 key
  atmost p block ptrs with (p-1) keys

- Every leaf node must be at same level

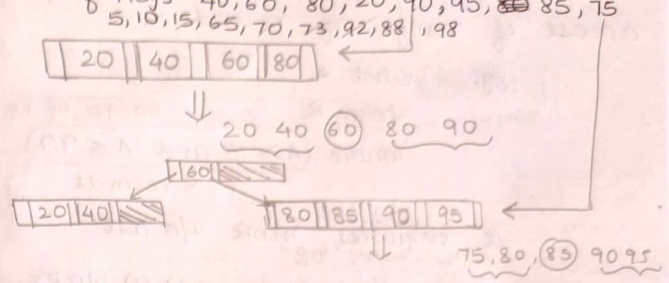
eg: order p: 15 (max Cp per node)



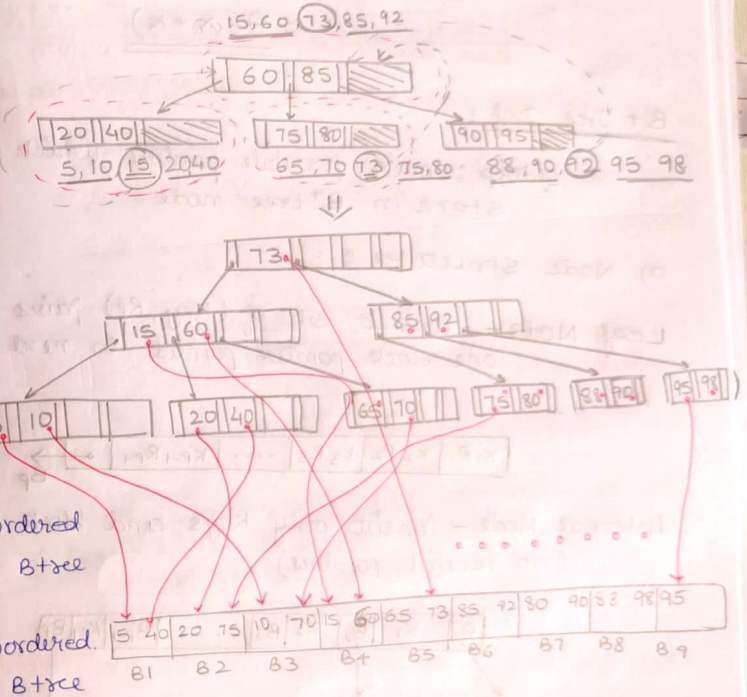


(min 50% occupied for all nodes except root)

Q- Construct B tree with order p : 5 (max cp per node) and sequence of keys 40, 60, 80, 20, 90, 95, 85, 75, 5, 10, 15, 65, 70, 73, 92, 88, 98



* next key insertion should be in appropriate leaf node



- if DB file ⇒ ordered then sparse B+tree index.
- if DB file ⇒ unordered then dense B+tree index.
  # of index = # of records

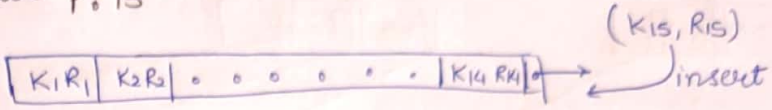
B Tree index best suitable for random access of any one record
eg: Select * from R where A=70; any one record access

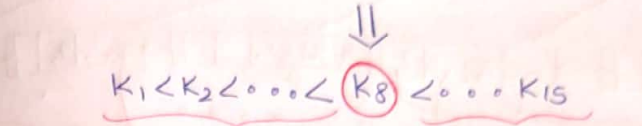
I/O cost : (k+1) blocks for random access of any one record.

(b) Balancing condition : same as B Trees.

eg:- Leaf splitting

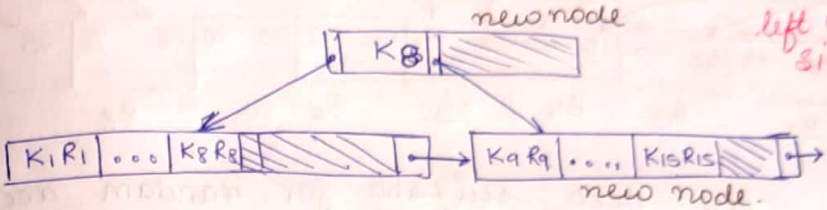
order p: 15



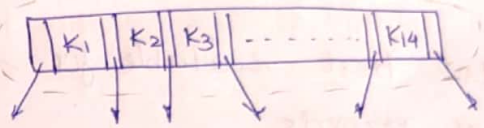


$K_1 < K_2 < \ldots < K_8 < \ldots K_{15}$

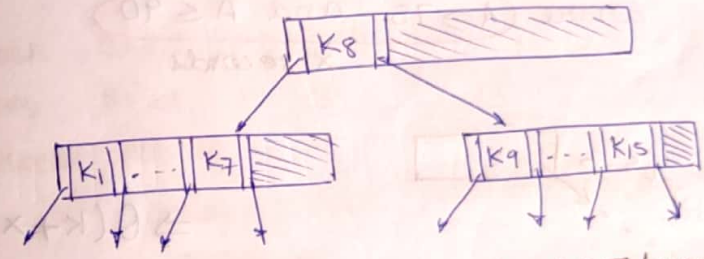
repitition on either left or right side



Internal node splitting



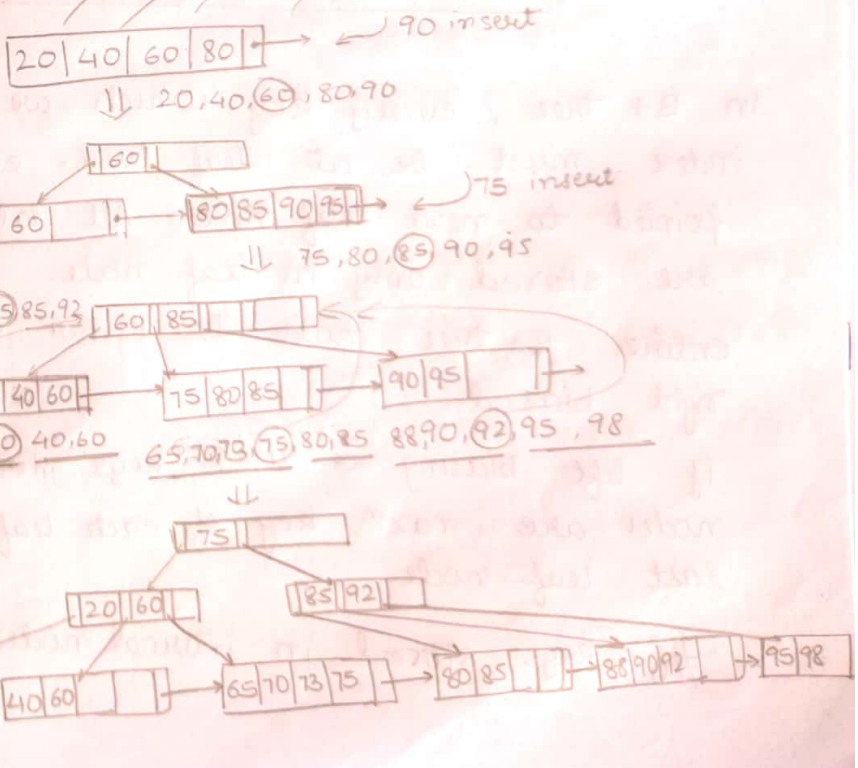
$K_1 < K_2 < \ldots < K_8 < \ldots K_{15}$



no more repitition

Q- Construct B+ tree with order p : 5 (max pointers per node)
sequence of keys : 40, 60, 80, 70, 90, 95, 85, 75, 5, 10, 15, 65, 70, 73, 92, 88, 98

Leaf node split



B+ Tree index best suitable for random access of any one record with $\overline{T/O\ cost}$: $(k+1)$ blocks

B+ Tree index also best suitable for sequential access of range of records

eg: select * from R
where $A \geq 75$ and $A \leq 90$
x records

$\Rightarrow (k+x)$ blocks

In B+ Tree, every key which we used for index must be at leaf level. each leaf node pointed to next leaf node. All data pointer are stored only in leaf node.

Entire B+ Tree must be either complete left or right biased.

If left biasing is used, keys stored in internal nodes are max^m key of each leaf node except last leaf node.

#f keys stored in internal node = #f leaf node - 1

**8- Find best possible order of B/B+ Tree node:

block size = 1024 B
search key = 10 B
block pointer = 12 B
Record pointer = 9 B

Order P: max possible block pointer in B tree node

What is best possible order of B tree node?

Block (B Tree) node	$P * Bp + (P-1) * (Rp+k)$	order P: $\max Bp \leq$ Block size

$$P * 12 + (P-1)(9+10) \leq 1024$$
$$12P + 19P - 19 \leq 1024$$
$$31P \leq 1043$$
$$P \leq \frac{1043}{31} \Rightarrow P = \left\lfloor \frac{1043}{31} \right\rfloor = 33$$

B- Block size = 512 B
key size = 8 B
Block pointer = Rp = 10 B

Order p: max possible keys can store in B tree node $\frac{2}{3}$

What is best possible order of B tree node?

Block (B tree) node	$(P*)*(K+Rp) + (P+1)Bp$	$\leq$ Block size

Order : max key/node

$$(P+1)10 + P(8+10) \leq 512$$
$$28P + 10 \leq 512$$
$$P = \left\lfloor \frac{502}{28} \right\rfloor = 17 \quad \text{// max keys store in B tree node}$$

Q- Block Size = 1024B
Search Key = 10B
B_p = 12B
R_p = 9B
Order p: max possible block pointers in B+ tree node
a what is best possible order of B+ tree
a) leaf node?
b) internal node?

a) $(p-1)*(K+R_p)+1*B_p \leq$ Block size
order p: max ptrs
$(p-1)R_p$ | 1 B_p
$(p-1)*(10+9)+12 \leq 1024$
$19p-19+12 \leq 1024$
$19p \leq 1031$
$p = \lfloor \frac{1031}{19} \rfloor = 54$

b) order p:
p B_p
$p*B_p+(p-1)K \leq$ Block size.
$(p-1)10+12p \leq 1024$
$p = \lfloor \frac{1034}{22} \rfloor = 47$

when p of leaf & internal node are same?
$(p-1)(K+R_p)+B_p = pB_p+(p-1)K$
$(p-1)K+(p-1)R_p = (p-1)B_p+(p-1)K$
$B_p = R_p$

NOTE: If size of R_p = size of B_p then B+ leaf node order equal to internal node order

Q- Block Size = 512B
search key = 8B
$B_p = R_p$ = 10B
Order p: max possible keys can store in B+ tree node
what is best possible order of B+ tree node?
a) leaf node
order p
p keys | 1 B_p
$p(K+R_p)+B_p \leq$ Block Size
$p(18)+10 \leq 512$
$p = \lfloor \frac{502}{18} \rfloor = 27$

b) internal node
(p+1) B_p
$p[K]+(p+1)B_p \leq$ Block size.
$8p+10p+10 \leq 512$
$18p \leq 502$
$p = \lfloor \frac{502}{18} \rfloor = 27$

Q- B: 1024 B
Key: 20 B
B_p: 15B
order p: max keys per node.
what is B+ tree best possible order.
internal node :- ∵ R_p is not given.
$pK+(p+1)B_p \leq 1024$
$20p+(p+1)15 \leq 1024$
$35p \leq 1009$
$p = \lfloor \frac{1009}{35} \rfloor = 28$

∴ Block size allocated for B Tree node = B+Tree node Block Size

then,

→ Order p: B Tree node < order p of B+Tree node

→ #s of B Tree nodes at level of B Tree nodes for n distinct keys > #s of B+Tree nodes at levels of B+Tree nodes for n distinct keys

→ (I/O cost) > (I/O cost)

Hence, B+tree is best for both range and random access queries.

Though B+tree is preffered than B tree for DB index because B+tree index access cost for both random access and range queries are less than B+tree index

8- find min, max keys and nodes in B tree/B+tree for given L levels and order p?

8- How many min keys/nodes in B/B+ tree order p:5 and level: 4?

order p: b/w ② to p bp for root

b/w $\lceil p/2 \rceil$ to p bp for other nodes

b/w $\lceil p/2 \rceil - 1$ to p-1 keys for any node

8- How many max keys/nodes in B/B+ tree order p:5 and level: 4?

order p: b/w 2 to p bp for root

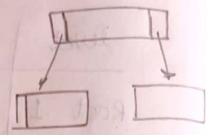
b/w $\lceil p/2 \rceil$ to p bp for other nodes

b/w $\lceil p/2 \rceil - 1$ to p-1 keys for any node.

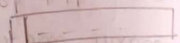
Q- How many B|B+ trees of order p:5 and level l:4 can have min key|nodes

Order p: b/w 1 to (2p) keys for root (10)
b/w p to 2p keys for otherwise.

level	min nodes	min Bp	min keys
root 1	1	2	1
2	2	2*6	2*5
3	2*6	12*6	12*5
leaf 4	72	-	72×5 → 360 B+
	87		431 → Btree



level	max nodes	max Bp	max key
root 1	1	11	10
2	11	11*11	11*10
3	121	121*11	121*10
leaf 4	1331	-	1331*10 = 13310 → B+
	1464		14640 → B



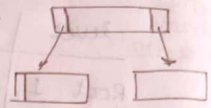
Order P(max bp per node)
height(h) : 0, 1, 2, ..., h

height(h)	min nodes	min Bp	min keys	max nodes	max Bp	max keys
0	1	2	1	1	P	P-1
1	2	$2\lceil p/2 \rceil$	$2(\lceil p/2 \rceil - 1)$	P	P^2	$P(P-1)$
2	$2\lceil p/2 \rceil$	$2\lceil p/2 \rceil^2$	$2\lceil p/2 \rceil(\lceil p/2 \rceil - 1)$	P^2	P^3	$P^2(P-1)$
3	$2\lceil p/2 \rceil^2$	$2\lceil p/2 \rceil^3$	$2\lceil p/2 \rceil^2(\lceil p/2 \rceil - 1)$	P^3	P^4	$P^3(P-1)$
⋮	⋮	⋮	⋮	⋮	⋮	⋮
h	$2\lceil p/2 \rceil^{h-1}$	-	$2\lceil p/2 \rceil^{h-1}(\lceil p/2 \rceil - 1)$	P^h	B-	$P^h(P-1)$
	$\frac{1+2(\lceil p/2 \rceil^h - 1)}{\lceil p/2 \rceil - 1}$		$\frac{1+2(\lceil p/2 \rceil - 1)(\lceil p/2 \rceil^h - 1)}{(\lceil p/2 \rceil - 1)}$	$\frac{P^{h+1}-1}{P-1}$		$P^{h+1}-1$
			$= 1+2(\lceil p/2 \rceil^h - 1)$			max B in B tree

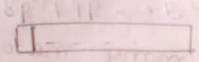
Q- How many B/B+ trees of order p:5 and level l:4 can have min key/nodes.

Order p: b/w 1 to (2p) keys for root (10)
b/w p to 2p keys for otherwise.

level	min nodes	min Bp	min keys
root 1	1	2	1
2	2	2*6 (+1)	2*5
3	2*6 (+1)	12*6	12*5
leaf 4	72	–	72×5 → 360 B+
	87		431 → B tree



level	max nodes	max Bp	max key
root 1	1	11	10
2	11	11*11	11*10
3	121	121*11	121*10
leaf 4	1331	–	1331*10 = 13310 → B+
	1464		14640 → B



Order P (max bp per node)
height (h) : 0, 1, 2, ..., h

height (h)	min nodes	min Bp	min keys	max nodes	max Bp	max keys
0	1	2	1	1	P	P-1
1	2	$2\lceil P/2\rceil$	$2(\lceil P/2\rceil-1)$	P^1	P^2	$P(P-1)$
2	$2\lceil P/2\rceil$	$2\lceil P/2\rceil^2$	$2\lceil P/2\rceil(\lceil P/2\rceil-1)$	P^2	P^3	$P^2(P-1)$
3	$2\lceil P/2\rceil^2$	$2\lceil P/2\rceil^3$	$2\lceil P/2\rceil^2(\lceil P/2\rceil-1)$	P^3	P^4	$P^3(P-1)$
⋮	⋮	⋮	⋮	⋮	⋮	⋮
h	$2\lceil P/2\rceil^{h-1}$	–	$2\lceil P/2\rceil^{h-1}(\lceil P/2\rceil-1)$	P^h	B–	$P^h(P-1)$
	$1+\frac{2(\lceil P/2\rceil^h-1)}{\lceil P/2\rceil-1}$		$1+\frac{2(\lceil P/2\rceil-1)(\lceil P/2\rceil^h-1)}{(\lceil P/2\rceil-1)}$	$\frac{P^{h+1}-1}{P-1}$		$P^{h+1}-1$
	$= 1+2(\lceil P/2\rceil^h-1)$					max keys in B trees

min keys : $\Theta((P/2)^h)$

max keys : $\Theta(P^h)$

Height of B trees with n nodes key and order p

$\Rightarrow 2\lceil P/2\rceil^h - 1 \le \text{keys} \le P^{h+1}-1$

Range of keys in B tree with order p && height h

$\Rightarrow 2\lceil P/2\rceil^{h-1}(\lceil P/2\rceil - 1) \le \text{keys} \le P^h(P-1)$

Range of keys in B+ tree

i) min height :- // Max fill factor each node or max keys per node

$n = P^{h+1}-1 \Rightarrow h = \boxed{\log_P(n+1) - 1}$

$\Theta(\log_P n)$

(ii) max height :- // Min Keys per node

$n = 2\lceil P/2\rceil^h - 1 \Rightarrow \boxed{h = \log_{P/2}\left(\frac{n+1}{2}\right)}$

$\Theta(\log_{P/2} n)$

$\Rightarrow \log_P(n+1) - 1 \le \text{height} \le \log_{P/2}\left(\frac{n+1}{2}\right)$

Range of height in B trees with order P

Q- Which is true about B Tree order P and height h

a) keys $\le 2\lceil P/2\rceil^h - 1$

b) keys $\ge P^{h+1}-1$

c) height $\ge \log_P(n+1)$

d) height $\ge \log_{P/2}\left(\frac{n+1}{2}\right)$

$\Theta\left(\left(\frac{P}{2}\right)^h\right) \le \text{Keys} \le \Theta(P^h)$

$\Theta(\log_P n) \le \text{height} \le \Theta(\log_{P/2} n)$

⇒ Bulk Loading B+ Tree :

[Construction of B+ tree in bottom-up approach]

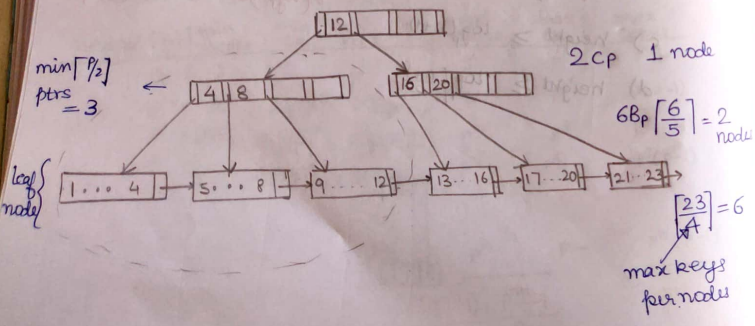
Design B+ tree leaf level to root level.

- Sort the keys which are used for B+ construction in ascending order.
- Design leaf nodes
  - → Distribute keys into leaf nodes
    result : # of leaf nodes
- Design Internal nodes
  - → if m nodes at level l
    then m child pointers required at level (l-1)
    Distribute m child pointers into nodes
    result : # of nodes at level (l-1)
- Repeat step 3 until root.

Construct bulk loading B+ Tree for keys [1,2,3,.... 23] and order p : 5 [max ~~child~~ pointers per node]

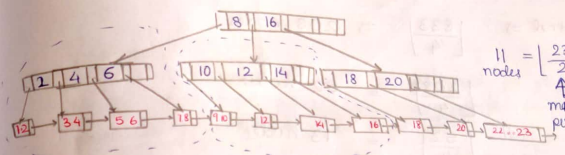
a) With min levels of index - or min index/ B+ tree nodes

max possible keys/ pointers per node
(4 keys & 5 ptrs per node)



min B+ Trees nodes : 9
min levels : 3

b) with max levels of index or max index/ B+ tree nodes
min possible keys/ pointers per node
[2 keys & 3 ptrs per node) min ⌈p/2⌉ = 3 ptrs



11 nodes = ⌊23/2⌋ keys
max keys per node

⌈11 Cp / 3⌉ = 3 nodes

no one should violate min condition
we can also take 3 4 4 ptrs
4 4 3
4 3 4
5 3 3

Q- Construct bulk loading B+ tree for keys [1, ... 2500] and order p : 7 (max ptrs)

a) How many min/max levels of B+ tree ?

(a) min levels :- max keys and pointer
(6 key, 7 ptr)

leaf nodes = ⌈2500/6⌉ = 417 nodes

Cp ⇒ Bp ⇒ ⌈417/7⌉ = 60 nodes ⇒ Cp 1st internal. min ⌈7/2⌉ ⇒ 4

⌈60/7⌉ ⇒ 9 nodes ⇒ 2nd

⌈9/7⌉ ⇒ 2 nodes ⇒ 3rd

⌈2/7⌉ ⇒ 1 node ⇒ 4th

489

min levels ⇒ 5

(b) max level :- min keys/ ~~nodes~~ ptrs (3 keys && 4 ptr)

min ⌈4/2⌉ = 2

leaf ⇒ $\lfloor \frac{2500}{3} \rfloor$ = 833 nodes

internal ⇒ $\lfloor \frac{833}{4} \rfloor$ ⇒ 208 nodes

$\lfloor \frac{208}{4} \rfloor$ = 52 nodes

$\lfloor \frac{52}{4} \rfloor$ = 13 nodes

$\lfloor \frac{13}{4} \rfloor$ = 3 nodes

1109

5+1 → root

max level ⇒ 6

NOTE: Bulk loading B+ Tree method can be used to find min/max levels of B+ Tree for given n-keys.

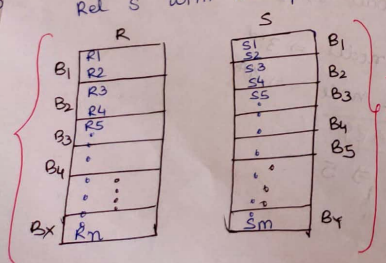
It is only applicable for B+ Tree.

JOIN ALGORITHMS

1. Nested loop Join Algo
2. Block Nested loop Join Algo

Rel R with n tuples occupies in X blocks

Rel S with m tuples occupies in Y blocks



Nested loop Join Algo :-

uses record no's for loop variable

R ⋈ S :
```
for(i=1; i≤n; i++)        // Record no. of outer rel
{
    for(j=1; j≤m; j++)    // Record no. of inner rel
    {
        Join { ith record of R, jth record of S }
    }
}
```

Disadv: if main memory space allocated for join is limited then, more access cost required to perform join using nested loop join algo.

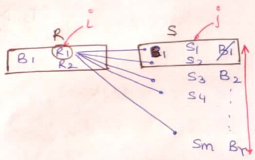
Block Nested loop Join Algo

R ⋈ S :
```
for(i=1; i≤x; i++)        // Block no of Outer Rel
{   for(j=1; j≤y; j++)    // Block no of inner Rel
    {
        Join { every record of ith block of R, jth block of S }
    }
}
```

⇒ Assume only two blocks of main memory allocated to join R & S

At any time one block of R area record and 1 block of S record can store in main memory.

a) R ⋈ S access cost using Nested loop join : only 2 block of MM



Y block access cost from S to join one record of R relation

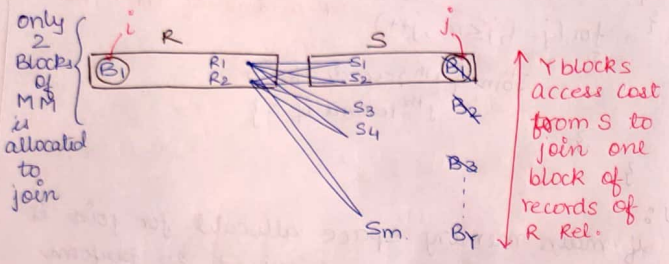
Total access time :

$(X + n * Y)$ blocks

if S is outer loop:

$(Y + m * X)$ blocks

b) R ⋈ S access cost using Block Nested loop Join Algo :



$(X + X * Y)$ blocks

Adv of Block Nested Algo over Nested loop Join :
less access cost if main memory space alloted is limited

Pg 69

Q18

T_1 : 2000 records
80 blocks

T_2 : 400 records
20 blocks

* Nest loop

$T_1 \bowtie T_2 \Rightarrow (X + n * Y)$
$= 80 + 2000 \times 20$
$\Rightarrow 40080$

$T_2 \bowtie T_1 \Rightarrow (Y + m * X)$
$= 20 + 400 \times 80$
32020 (c)

Q19 *Block

$T_1 \bowtie T_2 \Rightarrow (X + X * y)$
$= 80 + 80 \times 20$
$= 1680$

$T_1 \bowtie T_2 \Rightarrow (Y + y * X)$
$= 20 + 20 \times 80$
$= 1620$

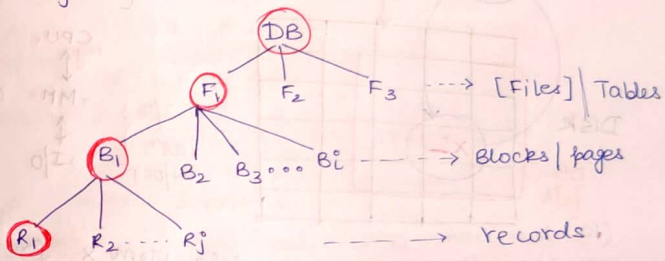
reduced ⇒ 32020
1620
30400 (b)

Q20 (a)

TRANSACTION AND CONCURRENCY CONTROL

Transaction :
set of logically related operations to perform unit of work

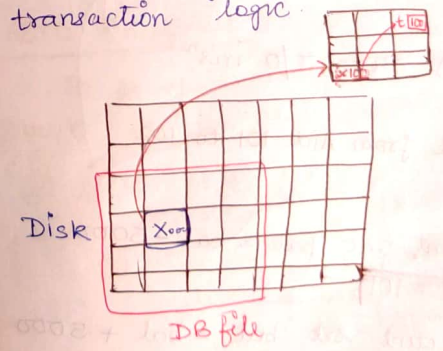
data item (Shared Resource) :
Data Base element which may require to access by many transaction



Data item : Record
Block
file
DB

Main operations in Transaction :

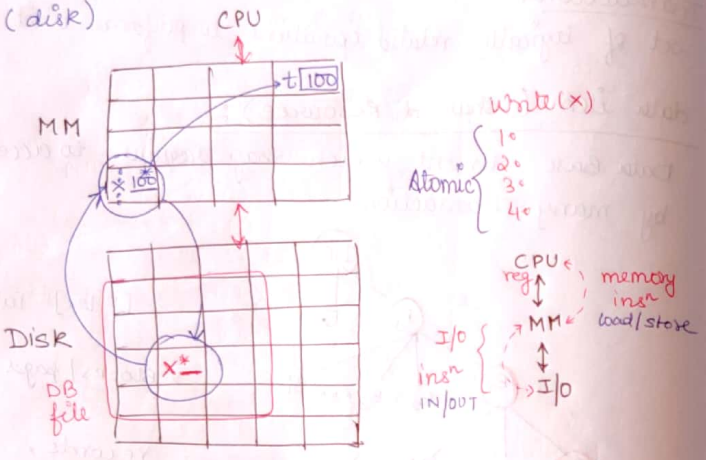
Read(X) :- [X is data item]
X is value of X from database file (disk) to main memory in order to use value of X in transaction logic



Atomic
1. Read(X)
2.
3.
⇓
no interruption
no context switch
without pre-emption

→ Find block which contain x from DB file
→ Transfer block to MM
→ Find address of data item x in MM block and copy value of x in programmed variable

Write (x): updation of data item x in DB file (disk)



→ find block which contain data item x from DB file
→ transfer block from disk to mm
→ find address of data item x and update value of x in main memory block.
→ replace modified block of MM into DB file (disk)

• Read (x) and write (x) are I/O ins^n

Trans : Transfer 5000 bal from Aid 101 to 102
(T1)

```
begin trans T1
    Update Account set bal = bal - 5000
    where aid = 101;
    Update Account set bal = bal + 5000
    where aid = 102;
    Commit ; // Transaction executed successfully
end Trans T1;
```

previous record value was required.

|| SQL Parser

```
begin Trans T1
    Read (bal of aid 101)
    bal = bal - 5000;
    Write (bal of aid 101)
    Read (bal of aid 102)
    bal = bal + 5000;
    Write (bal of aid 102)
    Commit
end trans T1
```

Set of R/W operation : Transaction

Trans : Set bal of 101, 102 aid's as 5000
(T2)

```
begin Trans T2
    Update Account set bal = 5000
    where aid = 101;
    Update Account set bal = 5000
    where aid = 102;
    Commit;
End Trans T2;
```

previous record value is not required

|| SQL parser

```
begin Trans T2
    Write (bal = 5000 where aid = 101)
    Write (bal = 5000 where aid = 102)
    Commit
end Trans T2
```

Blind Write

Trans (T3)

```
R(A)
R(B)
W(A)
W(C)
W(D)
Commit
```

W(C), W(D): Blind write

ACID Properties

To preserve integrity (correctness) transactions must satisfy acid property.

- A : Atomicity } maintained by Recovery management component of DBMS s/w (Theory basic)
- D : Durability }
- I : Isolation } maintained by Concurrency control component of DBMS s/w ***
- C : Consistency } maintained by USER DB Administrator / DB developer / end User

commit Rollback including commit

execute all operations of transaction or execute none of the operations of transaction before termination of transaction.

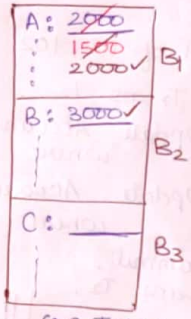
eg: Trans T_1 : Transfer 500 from A to B

begin Trans T_1
- R(A)
- A = A − 500
- W(A)
- * R(B)
- B = B + 500
- W(B)
- Commit

end Trans T_1

Rollback

failed



(DB Table)

DB file Correct

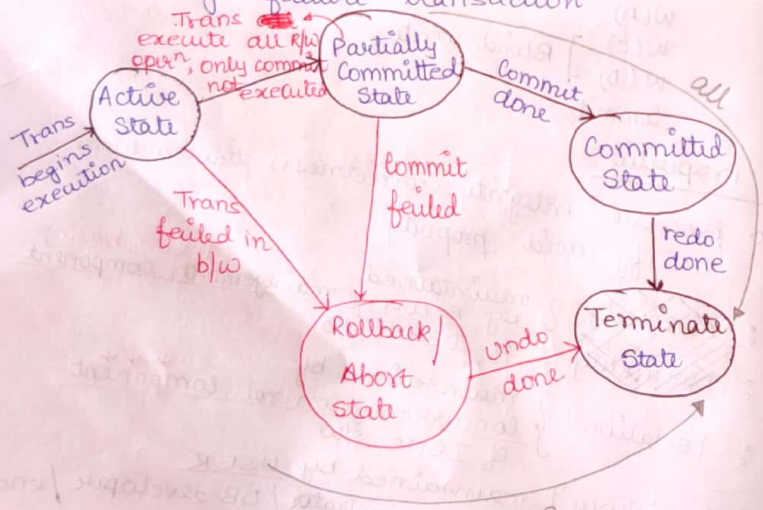
[Data ~~incorrect~~ becoz of atomicity voilation]

Recovery Management Component (RMC)

: (Abort) Roll backs transaction which is failed anywhere before commit.

Rollback (Abort):

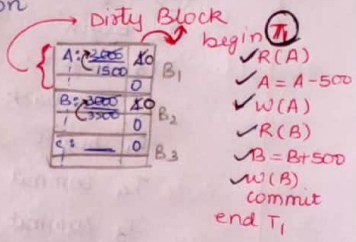
Undo all operations of DB file which are done by failure transaction



Transaction Log:

File maintained by RMC in disk to record all activities of each transaction

eg:
1. T_1 begins
2. T_1 R, A, 2000
3. T_1 W, A, 1000, 1500
4. T_1 R, B, 3000
5. T_1 W, B, 3000, 3500
6. failed



UNDO Operations of Trans

All updations of DB file because of Transaction failure (Rollback)

- Reset all W( ) to previous value
- Reset all dirty bit to 0
- Clear log entries

REDO Operations

All updations required in DB file because of Transaction commit and clean log entries of transaction

- Set dirty bits to 0
- Clear log entries
- Deallocate resources allocated to Trans

T_1 Commit → T_1 redo

T_2 Commit → T_2 redo

T_3 Commit → T_3 redo

T_4 Commit → T_4 undo

Overhead

if transaction → rarely then no problem., redo and undo can occur immediately but if transaction → frequently then, we cannot perform immediate redo & undo opern and this redo & undo becomes overhead. so we go for consistency check point to remove this overhead.

Consistency Check Point:

if consistency check point issues then,

- all committed transaction until previous check point performs redo
- all Trans of Rollback state until previous check point performs undo

9:00 AM start
- T_1 begin
- T_2 begin
- T_3 begin
- T_2 Rollback
- T_4 begin
- T_4 Commit
- T_3 commit

9:05 AM Check Point ⇒ T_3 T_4 redo } simultaneously
T_2 undo
- T_5 begin
- T_6 begin
- T_7 begin
- T_6 commit
- T_5 Commit

9:10 AM Check Point ⇒ T_5 T_6 redo
- T_8 begin
- T_8 commit

9:13 AM System Crash ⇒ T_8 redo } Crash recovery
T_1 T_7 undo

System Crash Recovery

- All committed transaction until previous checkpoint performs redo
- All uncommitted transaction in ~~whole~~ entire transaction system, change state to rollback and performs undo

8- Consider given log

1. T_1 begins
2. T_1 W, A, 0, 10
3. T_2 begins
4. T_3 begin
5. T_2 W, B, 5, 10
6. T_2 Commit
7. T_3 W B, 10, 50
8. Check Point ⇒ T_2 redo
9. T_4 begin
10. T_4 W, A, 10, 20
11. T_4 Commit
12. T_5 begin
13. T_5 W, C, 10, 50
14. T_3 Commit
15. System Crash ⇒ T_3, T_4 redo

What is Crash recovery?

a) T_2 T_3 T_4 redo, T_1 T_5 undo
b) T_3 T_4 redo, T_1 T_5 undo
c) T_2 redo, T_1 T_2 T_3 T_4 undo
d) T_2 T_3 redo, T_1 T_4 T_5 undo

Durability:

Transaction should able to recover under any case of failure and if failure occur during redo and undo, also should be able to recover.

Trans failure

recovery must then only durability satisfied

1. Power failure
2. S/W Crash
   - OS overloaded
   - DBMS overloaded
3. Concurrency Control protocol may kill transaction
4. Disk Crash (H/W Crash)

Log file && Trans state are stored in DISK

RAID Architecture
DISK Design

RAID: Redundant Array of Independent Disks

Consistency: [USER]

Logic of DB operations requested to perform transactions must be logically correct.

Trans: Transfer 500 from A to B

(A+B) before execution of T_1

equal

(A+B) after execution of T_1

T_1: Consistent

Isolation :

Concurrent execution of two or more transaction result must be equal to any result of serial execution of transactions.

Schedule

Time order execution sequence of two or more transactions

ex: S:

	T_1	T_2	T_3
1.	$r_1(A)$		
2.			$w_3(A)$
3.		$r_2(B)$	
4.	$w_1(A)$		
5.		$r_2(C)$	$w_3(B)$
6.			

↳ Serial Schedule :

- Trans must execute one after other

ex: T_1: Transfer 500 from A to B
T_2: Increment 20% to A & B

T_1	T_2
(S1) $r_1(A)$	$r_2(A)$ (S2)
$A = A - 500$	$w_2(A)$
$w_1(A)$	$r_2(B)$
$r_1(B)$	$w_2(B)$
$B = B + 500$	commit
$w_1(B)$	
commit	

T_2 reads A after T_1 writes → $T_1 : T_2$
T_2 reads B after T_1 writes → $T_1 : T_2$
T_1 reads A after T_2 writes → $T_2 : T_1$

T_1	T_2
$r_1(A)$	
$w_1(A)$	
$r_1(B)$	
$w_1(B)$	
commit	
	$r_2(A)$
	$A = A + A * 0.2$
	$w_2(A)$
	$r_2(B)$
	$B = B + B * 0.2$
	$w_2(B)$
	commit

- $T_1 \rightarrow T_2$ Serial Schedule
- $T_2 \rightarrow T_1$ Serial Schedule

Adv:
- No chance of incorrect result
- every serial schedule result always preserve integrity (correctness)

Dis:
- Degree of concurrency is very less

↳ Concurrent Schedule :

- simultaneous / concurrent / interleave execution of two or more transactions

ex:-

	T_1	T_2	(S3)
T_1 reads A before T_2 write → $T_2 : T_1$	$r_1(A)$		A: 2000
	$w_1(A)$		1500
T_2 reads B before T_1 write → $T_1 : T_2$		$r_2(A)$	1800
		$w_2(A)$	B: 3000
		$r_2(B)$	3500
		$w_2(B)$	4100
	$r_1(B)$		
	$w_1(B)$		
	C1		C2

(Non Serializable) Incorrect result

Not equal
$T_1 : T_2$ Serial
$T_2 : T_1$ Serial
↓
both satisfied then incorrect

	T_1	T_2
A: 2000 1500 1800	$r_1(A)$	
	$w_1(A)$	
		$r_2(A)$
		$w_2(A)$
A: 3000 3500 4200	$r_1(B)$	
	$w_1(B)$	
	C1	
		$r_2(B)$
		$w_2(B)$
		C2

Correct result equal to : $T_1 : T_2$ serial → satisfied correct

Degree of concurrency :
no of users can use DB simultaneously
- Throughput less
- resource utilisation not good
- Response time high

A: 2000 B: 3000

T_1	T_2	T_2	T_1
A = 1500	1800	A = 2400	1900
B = 3500	4200	B = 3000	4100
final		final	

Serial : Serializable

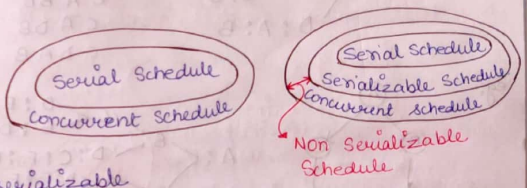
Adv:
- Throughput more
- resource utilisation is good
- Response time

Concurrency controller should not allow concurrent execution.
if concurrent execn result is incorrect.
It should allow those execn whose result is correct.

→ if concurrent execn result is equal to any one of serial result, then result of concurrent execn is correct
else it is incorrect.
⇒ Serializable if correct

Isolation also called Serializable Schedule

Serializable Schedule :
Schedule S is serializable if and only if S must be equal to some serial schedule



S_1, S_2, S_4 → serializable

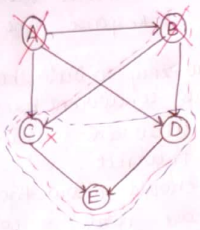
** Testing of Serializability :

↳ Conflict Serializable Schedule **

1. Topological order *
2. Conflict pair
3. Precedence Graph
4. Conflict Equal Schedule *

Topological Order :-
[Graph Traversal algo, only for Directed Acyclic Graph]
Graph G
1. Visit vertex(V) whose in-degree "0" and delete "V" from Graph
2. Repeat (1) for all vertices of Graph.

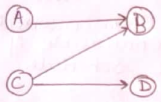
eg:



A : B < C : D : E
D : C : E

A B C D E
A B D C E
} Two topological orders

eg:

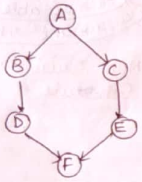


find # of topological orders.

A : C < B : D
D : B
C : A < B : D
D : B
D : A : B

A C B D
A C D B
C A B D
C A D B
C D A B
(5)

eg:

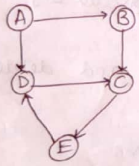


A : B < C < D : E : F
E : D : F
D : C : E : F
C < B < D : E : F
E : D : F
E : B : D : F
(6)

Q- # of topological orders for null Graph of n vertices
$= n!$

{ A B C D } $\Rightarrow 4!$ tological order.

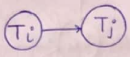
Q-



For cyclic Graph,
# of Tological orders = 0

Conflict Pair :-
pair of operations from schedule(S) is conflict pair iff,
1. Atleast one write
and 2. Over same data item
and 3. from different trans

eg: S : ..... $r_i(A)$ .... $w_j(A)$
S : ..... $w_i(A)$ .... $r_j(A)$
S : ..... $w_i(A)$ .... $w_j(A)$



Common: transaction must be different

Non-conflict Pair
eg: S : ..... $r_i(A)$ ... $r_j(A)$
S : ..... $r_i(A)/w_i(A)$ ..... $r_j(B)/w_j(B)$

S:

T_1	T_2
$r_1(A)$	
$r_1(B)$	
	$w_2(B)$
	$r_2(C)$
	$w_2(C)$

operation of same transaction neither conflict pair nor non-conflict pair

S:

T_1	T_2
$r_1(A)$	
	$w_2(B)$

$\Longleftrightarrow$ equal

T_1	T_2
	$w_2(B)$
$r_1(A)$	

S:

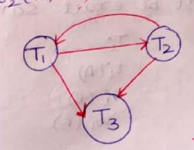
T_1	T_2
$r_1(A)$	
	$w_2(A)$

$\Longleftrightarrow$ not equal

T_1	T_2
	$w_2(A)$
$r_1(A)$	

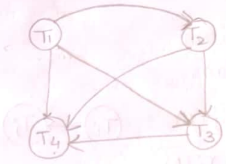
Precedence Graph :-
Graph G(V, E)
Vertices (V) : Transactions of schedule
Edges (E) : Conflict Pair precedence

eg: S: $r_1(A)$ $w_2(A)$ $r_2(B)$ $w_2(B)$ $w_3(B)$ $w_1(A)$ $w_3(A)$



Precedence Graph of Schedule (S)

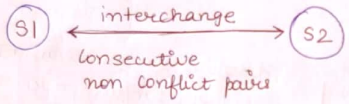
Q S: $r_1(A)$ $r_2(A)$ $r_3(A)$ $r_4(A)$ $w_1(B)$ $w_2(B)$ $w_3(B)$ $w_4(B)$



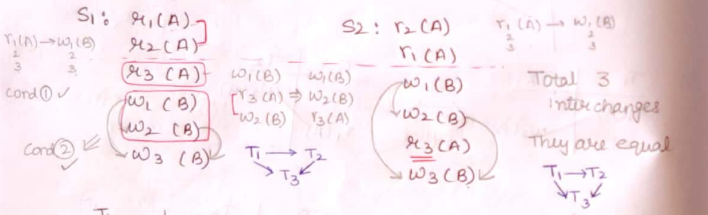
Conflict Equal Schedule :-

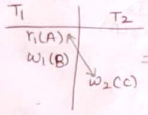
S1 S2 schedule Conflict Equal iff

- S_2 should derive by interchanging of consecutive non conflict pairs of S_1

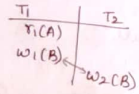


Q- Test S_1 S_2 conflict equal or not.





⇒ they are non-conflict pair but not consecutive ∴ we donot interchange as execn changes



⇒ result changes if conflict pair gets interchanged

S1 S2 Conflict equal iff

① each trans T_i of S_1 must be exact same trans in S_2

S_1 : ---- T_i ----

$r_i(A)$
$r_i(B)$
$w_i(B)$

S_2 : ---- T_i ----

$r_i(A)$
$r_i(B)$
$w_i(B)$

and ② every conflict pair precedence of S_1 must be same precedence in S_2

Q- S_1 : $r_1(A)$ $w_1(B)$ $r_2(A)$ $r_2(B)$ $r_1(B)$ $w_1(B)$ $T_1 \to T_2$ Not equal.

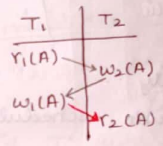
S_2 : $r_1(A)$ $w_1(A)$ $r_1(B)$ $w_1(B)$ $r_2(A)$ $r_2(B)$ $T_1 \to T_2$

if S_1, S_2 conflict equal, then precedence graph of S_1 S_2 is same

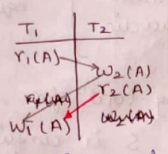
if S_1 S_2 with not same precedence graph then S_1 S_2 are also not conflict equal

if S_1 S_2 with same precedence graph then S_1 S_2 may/may not conflict equal.

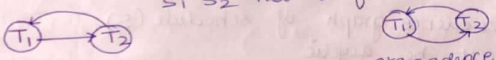
Q- S_1 :



S_2 :



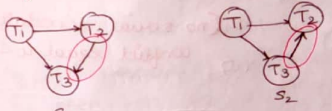
S_1 S_2 not conflict equal



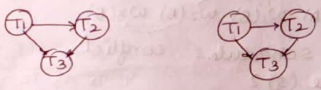
if S_1 S_2 schedules 1. with same precedence graph and 2. Acyclic Precedence graph.

then, S_1 S_2 are conflict equal

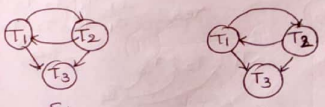
Conflict equal Testing using Precedence Graph :

1. 

S_1 S_2 not conflict equal

2. 

S_1 S_2 conflict equal.

3. 

S_1, S_2 may/may not conflict equal

Conflict Serializability

Given schedule S is conflict Serializable Schedule iff;

Some serial schedule S' must be conflict equal to some given schedule S.

{Given Schedule (S)} ⟺ (conflict equal) {S': Serial Schedule}

Given Schedule (S) — conflict serializable schedule

{Precedence Graph of S also must be acyclic} ⟺ (conflict equal) {S': Precedence Graph of serial Schedule is Acyclic}

Testing cond^n for Serializability

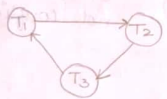
Schedule S is conflict serializable schedule iff, precedence graph of schedule (S) must be acyclic

and, conflict equal serial schedule of S are topological orders of precedence graph

eg:

S: $r_1(A)$ $w_2(A)$ $w_2(B)$ $r_3(B)$ $r_3(C)$ $w_1(C)$

Test S' conflict serializable

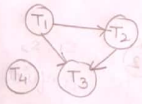


(no serial schedule conflict equal to S)

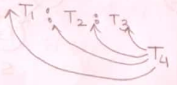
no

eg: S: $r_1(A)$ $r_2(A)$ $r_3(A)$ $r_4(A)$ $w_1(B)$ $w_2(B)$ $w_3(B)$

How many serial schedules conflict equal to Schedule (S) ?



Yes



T_1 T_2 T_3 T_4
T_1 T_2 T_4 T_3
T_4 T_1 T_2 T_3
T_4 T_4 T_2 T_3

(4) Serial schedules are conflict equal to Sche (S)

4 transaction means 4! Serial Schedule (24) out of which only 4 are conflict equal.

↳ View Serializable Schedule

Schedule S is view serializable schedule iff; some serial schedule S' must be view equal to given schedule S

{Given Schedule (S)} ⟺ (View equal) {S': Serial schedule}

View Serializable Schedule

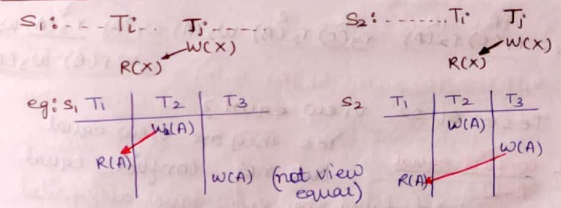
⇒ View equal schedule

S_1 S_2 Schedules view equal iff

(i) Initial Read

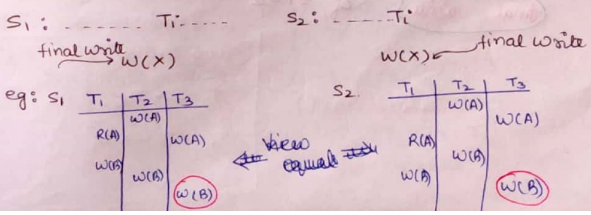
if T_i reads(X) from initial DB in S_1, then in S_2 also T_i should read(X) from initial DB

(ii) Updated Read



if T_i reads(X) which is updated by T_j in S_1 then in S_2 also T_i must read X updated by T_j only

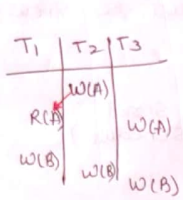
(iii) final write



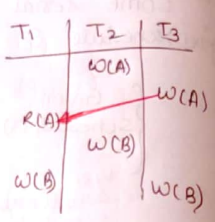
if T_i performs final write of data item X in S_1 then in S_2 also final write of X must be by T_i

Q- S1 S2 S3 Test view Equal

S_1:

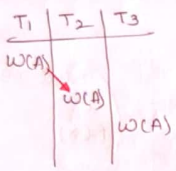


S_2:

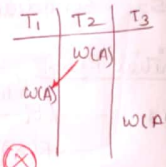


S1 S2 not equal

Q- S1:



S2:



Conflict equal ✗
View equal ✓

Q- S_1: $\overset{IR}{r_2(A)}\ \overset{IR}{r_2(B)}\ \overset{IR}{r_3(C)}\ \overset{IR}{r_1(C)}\ w_2(A)\ w_1(A)\ r_3(A)\ w_1(B)\ w_2(B)\ w_3(B)$

S_2: $\overset{IR}{r_1(C)}\ \overset{IR}{r_2(A)}\ \overset{IR}{r_3(C)}\ \overset{IR}{r_2(B)}\ w_2(A)\ w_1(A)\ r_3(A)\ w_2(B)\ w_1(B)\ w_3(B)$

a) Test S1 S2 view equal?

Yes they are view equal but not conflict equal.

S1 S2 every conflict equal
R-W, W-R → if failed then view equal also failed
W-W

conflict pairs of S1 S2 must be same precedence

⇓

S1 S2 every view equal
Initial Read
Updatid Read
final write
of S_1 S_2 must be same.

Testing of View Serializability :-

final Write :-

Given Sched(s)	View equal Serial (s')
X: T_i (T_j) final write	$T_i \rightarrow T_j$
Y: T_i T_j (T_k) final write	$T_i \rightarrow T_j \rightarrow T_k$, $T_j \rightarrow T_i \rightarrow T_k$] → $(T_i, T_j) \rightarrow T_k$
Z: (T_i) final write	every serial view equal (Any sequence)

Initial Read :-

Given Schedule(S)			View equal Serial (s')
data item	Initial Read	write	
X	T_i	T_j	$T_i \rightarrow T_j$
Y	(T_i T_j)	T_j T_k	$T_i \rightarrow T_j$ T_k, $T_j \rightarrow T_k$
Z	T_i	No other trans write(z)	every serial order view equal

Updated Read :-

Given Schedule(S)	View equal Serial (s')
a) $W_i(X) \rightarrow R_j(X)$ and T_K some other trans also writes (X)	$T_i \xrightarrow{T_K \times} T_j$ [T_i must before T_j && T_K should not be in b/w T_i & T_j]
b) $W_i(Y) \rightarrow R_j(Y)$ and no other trans writes (Y)	$T_i \rightarrow T_j$

Q- S: $R_1(A)\ W_3(A)\ W_1(A)\ W_2(A)$

test s is VSS?

Given Schedule(S)

FW: A: $T_3\ T_1$ (T_2)

IR:

Data	IRead	Write
A	T_1	$T_3\ T_1\ T_2$

View equal Serial (s')

$(T_3\ T_1) \rightarrow T_2$

$T_1 \rightarrow (T_3\ T_2)$

$S' : [T_1 : T_3 : T_2]$ Serial Schedule view equal to Sched (S)

S is View Serializable

S: $r_2(A)$ $r_2(B)$ $w_2(A)$ $w_2(C)$ $w_1(A)$ $r_1(C)$ $r_3(A)$ $w_1(B)$ $w_2(B)$ $w_3(B)$

Given schedule | View equal serial S'

IR: Data	IR	Write
A	T_2,	T_2 T_1
B	T_2	T_1 T_2 T_3
C		

FW: A: T_2 (T_1)
B: T_1 T_2 (T_3)
C: T_2

Update: (no other trans write C)
C: $T_2(W)$ $T_1(R)$
A: $T_1(W)$ $T_3(R)$
(other trans write A)

$T_2 \rightarrow (T_1)$
$T_2 \rightarrow (T_1, T_3)$
$T_2 \rightarrow T_1$
$(T_1, T_2) \rightarrow T_3$
any seq
$T_2 \rightarrow T_1$
$T_1 \xrightarrow{T_2^X} T_3$

S': $[T_2 \rightarrow T_1 \rightarrow T_3]$
Serial view equal to serial S
S is VSS

A: $T_2 \rightarrow T_1 \rightarrow T_3$
B: $T_2 \rightarrow T_1 \rightarrow T_3$
C: $T_2 \rightarrow T_1$

Q- S: $r_1(A)$ $r_2(A)$ $r_3(A)$ $r_4(A)$ $w_1(B)$ $w_2(B)$ $w_3(B)$ $w_4(B)$ $w_2(C)$ $r_4(C)$ $w_3(D)$. How many serial schedule VSS.

Given schedule | View equal serial S'

IR: Data	IR	Write
A	T_1, T_2, T_3, T_4	-
B		
B		

FW: A: —
B: T_1 T_2 T_3 (T_4)
C: T_2
D: T_3

Update: C: $T_2(W)$ $T_4(R)$

every serial order view equal.

$(T_1\ T_2\ T_3) \rightarrow T_4$ → 3!
any order
any order = 6
$T_2 \rightarrow T_4$

Pg 6+

(T1) S_3: $r_1(A)$ $r_3(D)$ $w_1(B)$ $r_2(B)$ $w_3(B)$ $r_4(B)$ $w_2(C)$ $r_5(C)$ $w_4(E)$ $r_5(E)$ $w_5(B)$

Given schedule | View equal S'

IR: DI	IR	W
A	T_1	-
B	~~T_2 T_4~~	~~T_2 T_3 T_5~~
C	~~T_5~~	~~T_2~~
D	T_3	-
E	~~T_5~~	T_4

FW: A: -
B: T_1 T_3 (T_5)
C: T_2
D: -
E: T_4

Update: B: $T_1(W)$ T_2, $T_3^{(W)}$ T_4
C: T_2 T_5
E: T_4 T_5

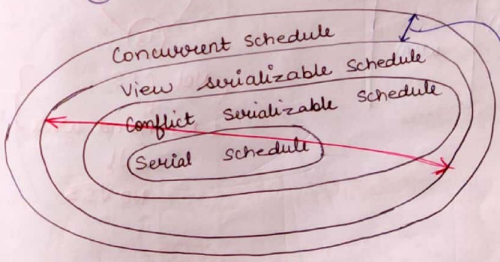
~~$(T_2\ T_4) \rightarrow (T_1\ T_3\ T_5)$~~
~~$T_5 \rightarrow T_2$~~
~~$T_2 \rightarrow T_4$~~
$(T_1, T_3) \rightarrow T_5$
$T_1 \rightarrow T_2$
$T_2 \rightarrow T_5$
$T_4 \rightarrow T_5$
$T_1 \times T_5^X$
$T_3 \rightarrow T_4$

T_1 T_2 T_3 T_4 T_5

T_1 : T_2 : T_3 : T_4 : T_5
T_3 : T_1 : T_4 : T_2 : T_5
T_3 : T_1 : T_2 : T_4 : T_5

① T_1 : T_2 : T_3 : T_4 : T_5
② T_3 : T_4 : T_1 : T_2 : T_5

⇒



Classification based on serializibility

Conflict serializable testing cond^n is only sufficient but not necessary cond^n for serializability cond^n.

CSS Testing
Precedence Graph of S is Acyclic
P Problem
$O(n^2)$
⟷ S is Serializable Schedule

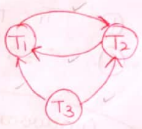
View Serializable Testing Cond^n is both sufficient and necessary condition for serializability testing

S satisfy View serializable cond^n
NPC Problem
$O(2^n)$
⟷ S is serializable Schedule

⇒ if schedule (S) not Conflict Serializable Schedule (CSS) and no blind write then,
{ Schedule (S) not View Serializable }

eg:

T_1	T_2	T_3
R(A)		
	R(A)	
		R(A)
W(A)		
	W(A)	



{ Not CSS + No Blind write }
⇓
NO VSS

Q] T_1 : $R_1(A)$ $R_1(B)$ $W_1(B)$
T_2 : $r_2(A)$ $r_2(B)$ $w_2(B)$
How many concurrent scheduled possible b/w T_1 T_2

① T_1: r(A) r(B) w(B) | T_2: r(A) r(B) w(B)
② T_1: r(A) r(B) w(B) | T_2: r(A) r(B) w(B)
③ T_1: r(A) r(B) w(B) | T_2: r(A) r(B) w(B)
④ T_1: r(A) r(B) w(B) | T_2: r(A) r(B) w(B)

S: T_1: r(A), r(B), w(A) | T_2: ○, ○, ○

$\frac{6!}{3!\,3!} = 20$ concurrent schedule

$^6C_3 * {}^3C_3 \Rightarrow \frac{6\times5\times4}{6} = 20$

Q] T_1 T_2 transactions with n, m operations each
How many concurrent schedule possible b/w T_1 T_2
$^{n+m}C_n \; {}^mC_m \Rightarrow \Rightarrow {}^{n+m}C_n = \frac{(n+m)!}{n!\,m!}$

Q7. $\frac{6!}{3!\,3!} = 20$ ⇒ ~~6×5×4~~ ⇒ 8 ~~contain non-serializable~~

Concurrent execution serializable

④ equal to T_1T_2 serial
S': T_1: r(A) r(B) w(B) | T_2: r(A) r(B) w(B)
Conflict equal

equal T_2T_1 serial
S': T_1: r(A) r(B) w(B) | T_2: r(A) r(B) w(B)
conflict equal

④ Serializable equal to T_2 T_1 serial

S': $r_1(A)$ $r_1(B)$ $W_1(B)$: $r_2(B)$ $W_2(B)$ $r_2(A)$
S': $r_2(A)$ $r_2(B)$ $W_2(B)$: $r_1(B)$ $W_1(B)$ $r_1(A)$

20 Concurrent

8 Serializable — 12 non serializable

2 serial — 6 non serial schedule which are serializable

Ans = (6)

26°

T_1: $r_1(A)$ $w_1(A)$ $r_1(B)$ $w_1(B)$

T_2: $w_2(B)$ $w_2(A)$

$\Rightarrow \frac{6!}{4!\,2!} \Rightarrow \frac{3 \times 5}{} = (15)$ concurrent

Serializable :

T_1	T_2
r(A)	
w(A)	
r(B)	
w(B)	
	w(B)
	w(A)

(2)

15

2 serializable — 3

2 serial — 0 non serial serialized

$r_1(A)$ $w_1(A)$ $r_2(B)$ $w_1(B)$ $w_2(B)$ $w_2(A)$

Ans : (0)

$w_1(A)$

B,14

T_1: $r_1(P)$ $r_1(Q)$ $w_1(Q)$

T_2: $r_2(Q)$ $r_2(P)$ $w_2(P)$

2 serial (b)

$\Rightarrow \frac{6!}{3!\,3!} = 20$

non-serial serializable = 0

Recoverability Classification :-

Concurrent execution may leads

- → Irrecoverable Problem
- → Cascading Rollback Problem
- → Lost update Problem

These problems can occur even if the schedule is serializable schedule.

Serializable cond^n not sufficient to avoid

- → Irrecoverable
- → Cascading Rollback
- → Lost update

Irrecoverable Schedule

Schedule S is irrecoverable iff,

T_j Reads data item X which is updated by T_i (and) commit of T_j before commit/rollback of T_i

eg :-

T_i	T_j
W(x)	
	R(x)
	commit
Commit/R	

IRRECOVERABLE

Rollback

T_1 : withdraw 5000 from 'A'

T_2 : check bal of 'A'

A : 20000

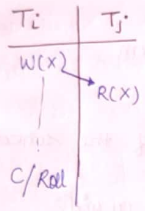
T_1	T_2
R(A)	
A = A − 5000	
W(A)	
	R(A)
	commit
*	

Sys Crash

IRREC

Result of T_2 incorrect & not possible to rollback because T_2 already committed

Uncommitted Read (Dirty Read)



Transaction T_j reads data item X which is updated by uncommitted trans T_i

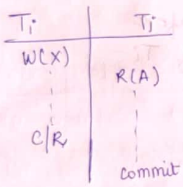
Recoverable Schedule

Schedule (S) recoverable iff

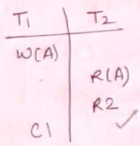
1] No uncommitted read in Schedule (S)

(OR)

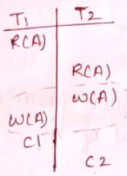
2] If T_j reads X which is updated by T_i then commit of T_j must delay until Commit / Rollback (C/ROll) of T_i



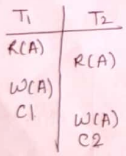
Q- Test given schedule recoverable / CLR / strict Recoverable

① 

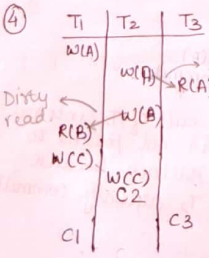
✓ × ×

② 

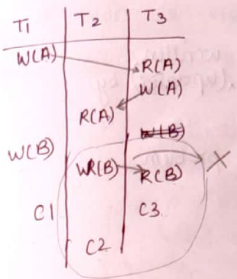
✓ ✓ ×

③ 

✓ ✓ ✓

④ 

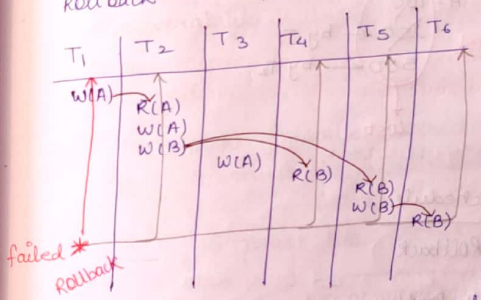
✓ × ×

⑤ 

Not × ×

Cascading Rollback Problem (Cascade Abort)

Failure of one transaction forced to rollback some set of other transaction (because of dirty read cascading Rollback occurs)

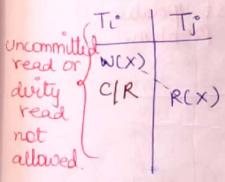


T_2 T_4 T_5 T_6 forced to roll back because of T_1 failure

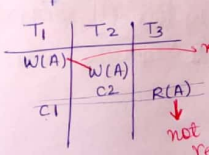
Disadv: Wastage of CPU execution time and I/O access cost and resource utilisation.

Cascadeless rollback Schedule (avoid cascade abort)

Dirty reads not allowed.



if T_i writes X, T_j read request of X which is updated by T_i must delay until C/R of T_i



if dirty read is present then schedule is Cascading Rollback Problem

Lost Update Problem :-

if T_j writes X which is already written (updated) by uncommitted transaction T_i
then it can cause lost update problem

T_i	T_j
W(X)	
⋮	W(X)

T_1	T_2
R(A)	
W(A)	
	W(A) ← Lost
	C_2

failed ✱

A = 100 → 200 ← by T_1 → 500 ← by T_2 (100 lost)

Strict Recoverable Schedule

Cascadeless Rollback
(and)
No Lost Update Problem

T_i	T_j
W(X)	
C/R	
	R(X)

done to avoid Irrecoverable and Cascading Rollback problem.

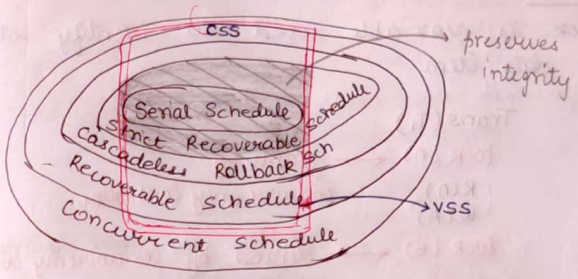
T_i	T_j
W(X)	
C/R	
	W(X)

done to avoid lost update problem.

Strict Recoverable

T_i	T_j
W(X)	
C/R	
	W(X) / R(X)

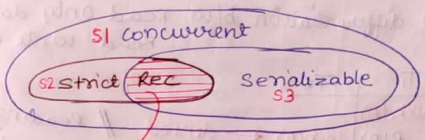
if T_i writes X then read of X or write of X of T_j must delay until commit or roll back of T_i



classification based on Recoverability :

both recoverability and serializability classification are independent of each other.

Schedule(S) Preserves correctness (integrity)
iff S must be serializable
(and)
S must be strict recoverable



Concurrency Control Protocols Goal :

Concurrency control Protocol should not allow to execute if

→ S not serializable
(or)
→ S non strict recoverable

1. Locking Protocol
2. Time stamp ordering Protocol

Locking Protocol :-

Lock:

Lock is variable used to identify status of data item.

Trans (T_i)
- lock (A) ← grants by concurrency Controller
- R(A)
- W(A)
- lock (B) ← denied by Concurrency Controller
- : wait for data item
- lock (B) ← grants by Concurrency Controller
- W(B)
- Unlock (A)
- Unlock (B)

Lock is used to synchronize the transaction

Binary Lock

→ lock
→ unlock

No differentiation b/w Read only data
Read write data

T_1	T_2
lock (A)	
R(A)	lock(A) ← denied // Reading is not allowed simultaneously it is done for consistency
	R(A)

Disadv: less degree of concurrency.
Unable to differentiate Read only data with Read write data so that simultaneous read oper^n also not allowed. which leads to less degree of concurrency.

Shared – Exclusive locking:

Shared lock (S) Read only lock
Exclusive lock (X) Read / write lock

T_1	T_2
S(A)	
R(A)	
	S(A) ← Request granted.
	R(A)

Simultaneous Read oper^n is now allowed. as it doesnot cause inconsistency. So more degree of concurrency.

T_1	T_2
S(A)	
	X(A) ← Request denied.

lock compatible table:

Requesting by T_j \ Hold by T_i	S	X
S	Yes	NO
X	NO	NO

Inc(A)
A=A+1

Inc(A)
A←A+1

→ Increment lock, it is granted as it doesnot cause inconsistency

Two phase locking Protocol :-

[Guaranteed Serializable]

Trans (T)
- X(A)
- S(B)
- X(C) — Locking Phase // User should collect all required data item before unlocking.
- X(D) — lock point of T
- U(C) ← first unlock // no more request, allowed
- U(A)
- U(D) — Unlocking Phase [shrinking Phase]
- U(B)

Transaction (T) allowed to request lock on any data item in any mode until first unlock oper^n of Trans (T) [lock request not allowed in unlocking phase]

ex: Trans (T_1)
Transfer 500 from A to B

T_1
- X(A)
- R(A)
- W(A)
- X(B)
- U(A)
- R(B)
- W(B)
- U(B)

2PL

eg:- Trans (T_2)
display Total balance of A, B

T_2
- S(A)
- R(A)
- S(B)
- U(A)
- R(B)
- U(B)
- Commit

2PL

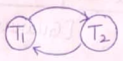
if T_1, T_2 executed concurrently, then guaranteed serializability

Concurrent execution of T_1 T_2 guaranteed serializable schedule.

If schedule (S) not conflict serializable schedule then schedule (S) not allowed by 2PL protocol.

eg- S:

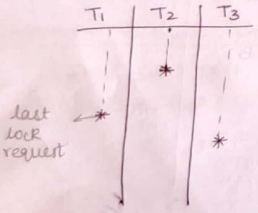
T_1	T_2
X(A)	
R(A)	
X(B) W(A)	
U(A)	
	S(A) ✓
	R(A)
	S(B) ← denied because B is not unlocked by T_1
	R(B)
R(B)	
W(B)	
U(B)	



CSS ✗
2PL ✗

If schedule (S) allowed to execute by 2PL then Schedule (S) guaranteed Conflict serializable schedule and conflict equal serial orders are lock point orders (topological order, lock point order are same)

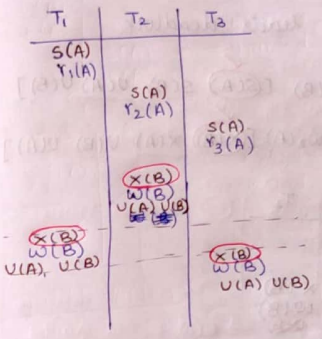
⇒ S allowed by 2PL ⟹ S is CSS
Conflict equal serial Sched: lock Points order



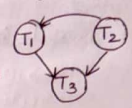
If S allowed by 2PL then S is CSS and equal serial

$T_2 : T_1 : T_3$

eg:- S: $r_1(A)$ $r_2(A)$ $r_3(A)$ $w_2(B)$ $w_1(B)$ $w_3(B)$



- Allowed by 2PL
- CSS



$T_2 : T_1 : T_3$

- lock Point Order = Serial Schedule Order
  T_2 T_1 T_3

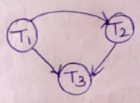
Every 2PL schedule is always Conflict serializable sched - but not every CSS can allowed to execute by 2PL

2PL schedule
Conflict serializable

Q: S: $w_1(A)$ $w_2(A)$ $w_3(A)$ $w_1(B)$ $w_2(B)$ $w_3(B)$
Which is true?

(a) S is CSS and in 2PL
(b) S is CSS and not in 2PL
(c) S is not CSS and in 2PL ← never
(d) S is not CSS && not in 2PL

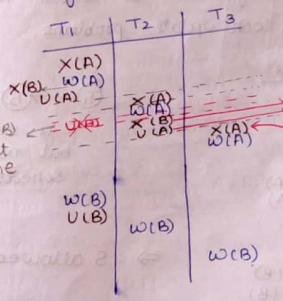
CSS:



✓

If Cycle then donot check 2PL
∵ CSS(✗) → 2PL ✗

2PL Testing:



Limitations of 2PL restriction

1] 2PL restriction may leads deadlock

T_1 : $r_1(A)$ $r_1(B)$ [S(A) S(B) U(A) U(B)]

T_2 : $w_2(B)$ $w_2(A)$ [X(B) X(A) U(B) U(A)]

T_1	T_2
S(A) R(A)	
	X(B) W(B) U(B)
denied → S(B) R(B) (T_1 waiting for T_2)	
	X(A) W(A) ← denied, T_2 waiting for T_1

2] 2PL restriction may leads starvation

T_1	T_2	T_3	T_4	T_5 ...
	S(A)			
denied bec of T_2 → X(A)	U(A)	S(A)		
denied bec of T_3 → X(A)		U(A)	S(A)	
denied bec of T_4 → X(A)				

Starvation

3] 2PL condition not sufficient to avoid IRREC problem, Cascading Rollback, lock update problems.

S:

T_1	T_2
X(A) W(A) X(B) U(A)	
	S(A) R(A)
W(B) U(B)	
	X(B) W(B) U(A) U(B) C2
C1	

$T_1 \rightarrow T_2$ CSS ✓

S is serializable but not recoverable Schedule

⇒ S allowed by 2PL

basic 2PL :- lock requests of T not allowed in Unlocking phase of T

(and) Strict Rec : All exclusive lock of Trans(T) must hold until C/R of trans(T)

T_i	T_2
X(A) W(A)	
⋮	
C/R U(A)	R(A)/W(B)

X(A) must hold until C/R, i.e, no other trans grants for S(A)/X(A)

Strict 2PL schedule
- ↳ guarenteed conflict serializable
- ↳ Guarenteed strict Recoverable
- ↳ Deadlock & starvation are possible.

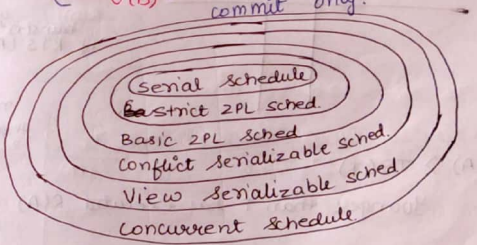
Ex of S/X and strict 2PL Trans

Trans(T)

Locking Phase:
S(A)
R(A)
X(B)
W(B)
S(C)
R(C)
X(D)

Unlocking Phase:
W(D)
U(A)
U(C)
] – shared lock can release before/after commit

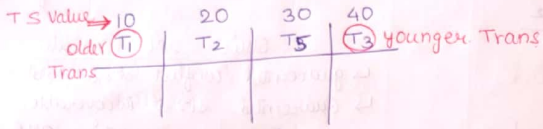
Commit
U(B)
U(D)
] – exclusive lock should release after commit only.



Time Stamp Ordering Protocols :-

Time stamp of Transaction :

Unique value assigned by DBMS for each transaction in sequence order (Ascending Order)



TS value of transaction can be used as
↳ Transaction ID
and ↳ also can used to set priority

it also uses TS value of data item :

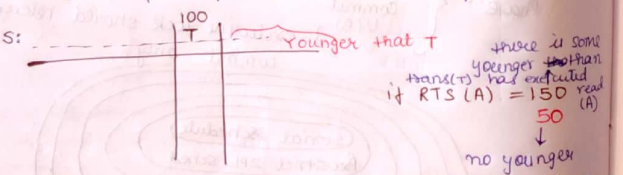
Read TS(x): Highest trans TS value that has performed R(X) successfully

Write TS(x): Highest Trans TS value that has executed W(X) successfully.

eg:-

RTS (A) ~~10~~ 30

WTS(A) = ~~10~~ 40



there is some younger than T has executed read (A)
if RTS(A) = 150
50
↓
no younger than T performs read

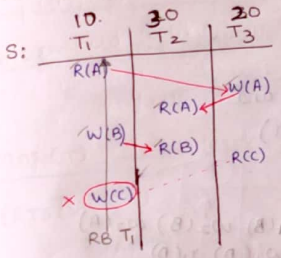
if RTS(A) > TS(T)
Younger than T has executed R(A)

else
No younger than T has reads A

Basic Idea of TS Ordering Protocol :-

Concurrent execution of two or more transaction result must be serial schedule result based on time stamp Ordering. then TS ordering protocols are allowed to execute protocols.

If Read/Write request of Trans voilates equal serial schedule condition based on TS orders then Roll back transaction.



equal to serial based on TS orders

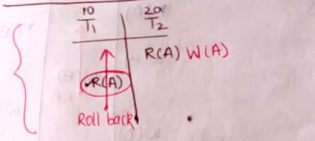
S: equal to serial
10 20 30
T1 : T3 : T2

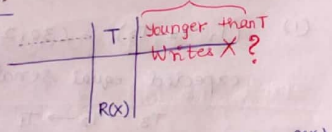
Conflict pair which are allowed
10 20 20 30 10 30
T1 → T3 T3 → T2 T1 → T2
allowed.

if $W_2(A)\ R_1(A) \Rightarrow T_2 \rightarrow T_1$ (not allowed)
∴ it is not equal to serial.

Basic TS Ordering Protocol :

Trans(T) issues R(X) :-

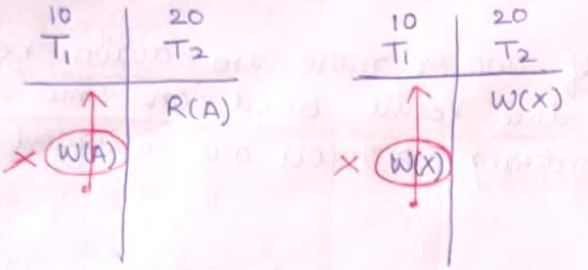




if younger T W(X) then R(X) roll back else allowed.

```
if (WTS(X) > TS(T))
{
    Rollback T
}
else
{
    allowed R(X) by T and
    Set RTS(X) = max{RTS(X), TS(T)}
}
```

Trans T issues W(X):-



CSS: $T_2 \rightarrow T_1$

TS: $T_1 \rightarrow T_2$ → not match not allowed.

if (RTS(X) > TS(T))
    Rollback T
else if (WTS(X) > TS(T))
    Rollback T
else
{
    allowed W(X) by T and
    set WTS(X) = TS(T)
}

Q:

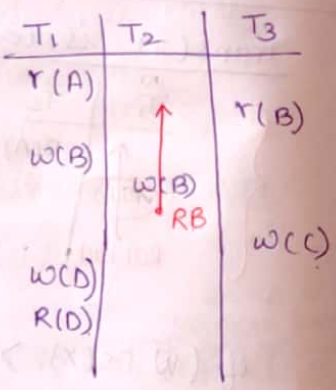
S: $r_1(A)$ ~~$r_2(A)$~~ $r_3(B)$ $w_1(B)$ $w_2(B)$ $w_2(A)$
$w_3(C)$ $w_2(C)$ $w_1(D)$ $r_1(D)$

Which Trans RB using BTS Protocol with TS value.

(i) $(T_1\ T_2\ T_3) = (30, 20, 10)$

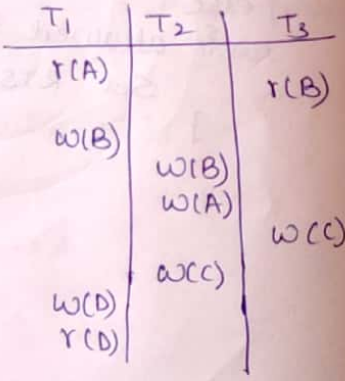
expected equal serial.

$T_3 \rightarrow T_2 \rightarrow T_1$
10 20 30



(ii) $(T_1\ T_2\ T_3) = (20\ 30\ 10)$

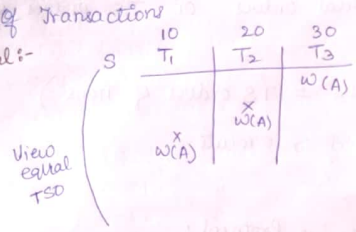
expected equal serial
$T_3 \rightarrow T_1 \rightarrow T_2$



NO Rollback.

Schedule which is executed by TW TSOP result view equal serial schedule based on TS order of Transactions

Goal:-

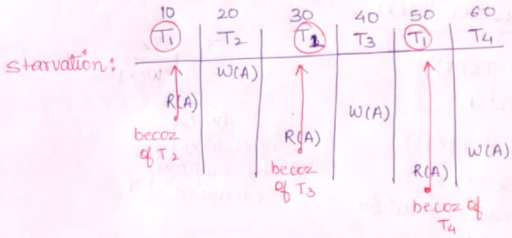


BTSO/ TWRTS

→ Guareented serializable

→ free from Deadlocks

→ Not free starvation



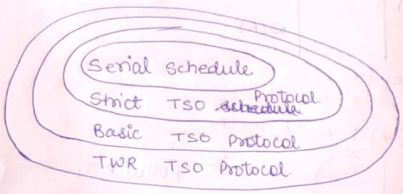
→ Not guareented for strict recoverable

Strict TS Ordering Protocol:

1] Basic TSO/TWRTS for serializability

and

2] strict recoverable must satisfy



Tuple Relational Calculus (TRC)

⇒ Non Procedural Query language
What data retrieve

⇒ Uses first Order logic and Predicate calculus formulas

Basic formulas

P, Q Predicati (simple stmt which can T/F)

$P \vee Q$

$P \wedge Q$

$\neg P$

$P \rightarrow Q$

$X \in Rel$ where X is variable

quantifier:

$\exists X \in Rel\ (P(X))$

TRUE if atleast one record (X) of rel satisfy P(X)

$\forall X \in Rel\ (P(X))$

TRUE if every record (X) of rel satisfy P(X)

Tuple Variables

free tuple Variable

$T \in stud$

stud

T: Universe

Bounded Tuple Variable

$\exists T_1 \in stud\ (P(T_1))$

$\forall T_2 \in course\ (P(T_2))$

Variable bounded by quantifier

Format of TRC Query

$\{T \mid P(T)\}$:-

Result Variable

T: Tuple Variable

P(T): Formula over tuple variable

Retrieves set of Tuples(T) those are satisfied formula P(T)

eg $\{T \mid \underbrace{T \in stud \wedge T.age > 20}_{P(T)}\}$

$\equiv \sigma_{age>20}(stud)$

Retrieves students whose age more than 20.

Q- Stud (Sid Sname Age)

Course (Cid Cname Instructor)

Enroll (Sid Cid fee)

⇒ Retrieve sname's enrolled ∃ some ∃ course

Stud (Sid	Sname)		Enroll (Sid	Cid)
∃T1	S1	A	∃T2 S1	C1
	S2	B	S1	C2
	S3	C	S2	C2

(for some T_1, there should be one T_2 which satisfy the cond^n)

$\{ T / \exists T_1 \in Stud \ \exists T_2 \in Enroll$

$(T_1.Sid = T_2.Sid) \wedge (T = T_1.Sname) \}$

(or)

$\{ T.Sname / (T \in Stud) \wedge (\exists T_1 \in Enroll (T_1.Sid = T.Sid)) \}$

bec this is used in result

always true

⇒ Retrieve sid's enrolled ∃ some course taught by Korth

~~$\exists T_1 \in Stud \quad \exists T_2 \in$~~

Enroll (Sid Cid) = course (Cid Inst) Korth

$\exists T_1$ $\exists T_2$

RA: π_{sid} (Enroll ⋈ Course) Ins = Korth

RC: $\{T / \exists T_1 \in Enroll \ \exists T_2 \in Course.$

$(T_1.Cid = T_2.Cid \wedge T_2.Inst = Korth \wedge T = T_1.Sid)\}$

⇒

Set operations :-

R & S relations

$R \cup S = \{ x \mid x \in R \vee x \in S \}$ OR

$R \cap S = \{ x \mid x \in R \wedge x \in S \}$ and

$R - S = \{ x \mid x \in R \wedge \neg (x \in S) \}$

but not

⇒ Retrieve sid's whose age more than 20 or enrolled some Korth course

RA: $\pi_{sid}(\sigma_{age>20}(Stud)) \cup \pi_{sid}(\sigma_{E.Cid = C.Cid \wedge C.Inst = Korth}(Enroll \ldots))$

RC: $\{T / \exists T_1 \in Stud (T_1.age > 20 \wedge (T = T_1.Sid) \vee$

$\exists T_1 \in Enroll \ \exists T_2 \in Course$

$(T_1.Cid = T_2.Cid \wedge T_2.Inst = Korth \wedge (T = T_1.Sid))\}$

⇒ $\{T / \exists T_1 \in Stud (T = T_1.Sid) \wedge$

$\neg (\exists T_1 \in Stud \ \exists T_2 \in Stud ((T_1.age < T_2.age) \wedge (T = T_1.Sid)))\}$

age ≥ every age

$\{T / \exists T_1 \in Stud \ \forall T_2 \in Stud \ (T_1.age \geq T_2.age \wedge T = T_1.Sid)\}$

retrieves sid's whose age is maximum.

$P \wedge \neg Q = P - Q$

⇒ $\{T / \exists T_1 \in Enroll (T = T_1.sid) \wedge$

$\neg (\exists T_1 \in Enroll \ \exists T_2 \in Enroll (T_1.sid = T_2.sid \wedge T_1.cid \neq T_2.cid \wedge T = T_1.sid))\}$

Sid's enrolled exactly one course

$\pi_{sid}(E) - \pi_{sid}(\sigma_{T_1.sid = T_2.sid \wedge T_1.cid \neq T_2.cid}(T_1 \times T_2))$

Conclusion

***Unsafe TRC Query :-**

TRC Query with infinite set of records in result because TRC uses free variable for result and boundary of free variable is infinite.

eg:- $\{T \mid \neg(T \in Stud)\}$

result : Set of Tuples T those are not belongs to stud relation
[Infinite record set in result]

{Expressive Power of safe TRC Queries} ≡ (equal to) {expressive Power of Basic RA Queries}

Basic RA Queries :-

- Queries which can formulate using
  $\{\pi, \sigma, \rho, \cup, -, \times$ (fundamental) $\cap, \bowtie, \bowtie_c, \exists\bowtie, \bowtie\!\!\!\!-, \div\}$ (derived)
- Queries failed to express using Basic RA also fails in Safe TRC

  → Count of records
  Count Attribute Values
  Sum of Attribute Values
  Avg of Attribute Values

  → Ordering of records in Asc/Desc etc