04/11/20

Database Management System (8M)

Database Design:

1. E-R model
2. Relational model
3. Functional dependency, Normalization.

Implementation:

4. Relational Algebra, Relational Calculus
5. SQL

Maintainance:

6. Transaction Management
7. File Organization
   Indexing

Data: Data is a raw fact (Abstract in nature).

Information: Meaningful / processed data.

Data is stored in the form of records

Records:

Collection of logically related data.

Database:

Collection of records

Management:

It is done using set of programs.

} ⇒ DBMS

DBMS:

Data Structure Used	DBMS
Graphs	Network DBMS
Trees	Hierarchial DBMS
Tables / set	Relational DBMS (RDBMS)
object	object oriented DBMS (OODBMS)
objects & tables	object - Relational DBMS (ORDBMS)

Examples of RDBMS

ORACLE, DB-2, SQL-Server, MYSQL, Sybase

Logical Database Design using Relational model:

→ Relation consists of records (tuple).

Each record has certain no of attributes

→ Every relation is described using 2 things.

(i) Relation schema

(ii) Relation instance

* Relation schema specifies structure of relation.

Relation schema specifies name of relation, attributes, domain of attribute, conditions on attributes (Eg: primary key)

↓ constraints

* Relation instance specifies the tuples present at a particular time.

Degree (or arity) of relation: It is no of columns i.e., no of attributes.

Cardinality of relation: It is no of tuples.

Integrity Constraint: It is a condition specified on a database schema.

There are 5 types of integrity constraints:

Table constraints (i.e. they are specified on 1 table):
- (i) Primary key (unique, not null)
- (ii) Unique (no duplicates but null is allowed)
- (iii) not null
- (iv) Check (User defined conditions) (Eg: age >= 18)

Referential integrity constraint:
- (v) Foreign key (Used to show relation blw 2 tables)

key Constraint:

key is a set of attributes which uniquely determines each tuple in a relation.

There are 3 types of key constraints:

i) Candidate key:

It is minimal set of attributes which uniques identifies each tuple in a relation.

i.e., If S is candidate key then no proper subset of S is a key.

Eg: Consider

student (Roll_no, name, father_name, passport_no)

Here

(Roll_no) ← key

(name, father_name) may or may not be key

(Passport_No) may or may not be key

because some students may not have passport.

(Roll_no, name) ← key

(Roll_no, passport_no) ← key

Now

(Roll_no) ← candidate key

(Roll_no, name) is not a candidate key

(ii) Super key:

A set of attributes which uniquely identifies a tuple in a relation.

(or)

Set of attributes to which a candidate key is subset

(or)

Superset of a candidate key

Eg: (Roll_no) ← Super key

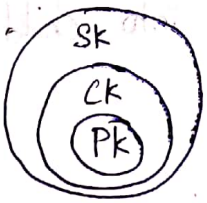
(Roll_no, name) ← Super key.

→ Every candidate key is a super key but reverse need not to be true.

(iii) Primary key:

Among various no of ~~primer~~ candidate keys, we choose one ck as primary key.

But general design convention ck will less number of attributes is choosen as primary key.

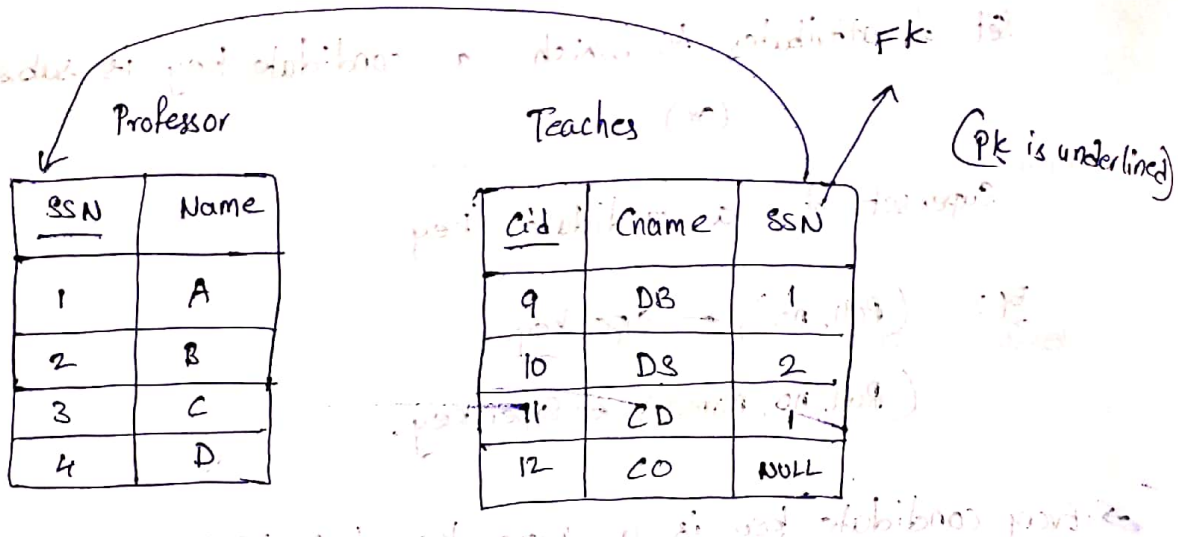


Whenever it is mentioned key, we consider candidate key

Foreign key Constraints (Referential integrity Constraint) :-

Foreign key is set attributes used to refer set of attributes of another relation or same relation.

Eg:



SSN	Name
1	A
2	B
3	C
4	D

Cid	Cname	SSN
9	DB	1
10	DS	2
11	CD	1
12	CO	NULL

→ Fk may have both duplicates & null values.

→ Here Teaches is called Referencing relation (Child table)
→ Here Professor is called Referenced relation (Parent table)
} Parent Child relationship

→ Every value under Fk column of the child table must be present in under Pk column of the parent table. But reverse need not to be true. (NULL is not considered a value, NULL is just value's absence)

→ Deletion and updation from parent, Insertion into child may cause violations of above rule.

on delete cascade / on update cascade:

→ It is a condition set on Fk.

when a violation of deletion/updation occurs in parent table, the corresponding rows of child table are deleted/updated.

Set null on deletion violation:

→ when this condition is set on Fk of child table, the Fk values of corresponding rows are set to NULL on deletion.

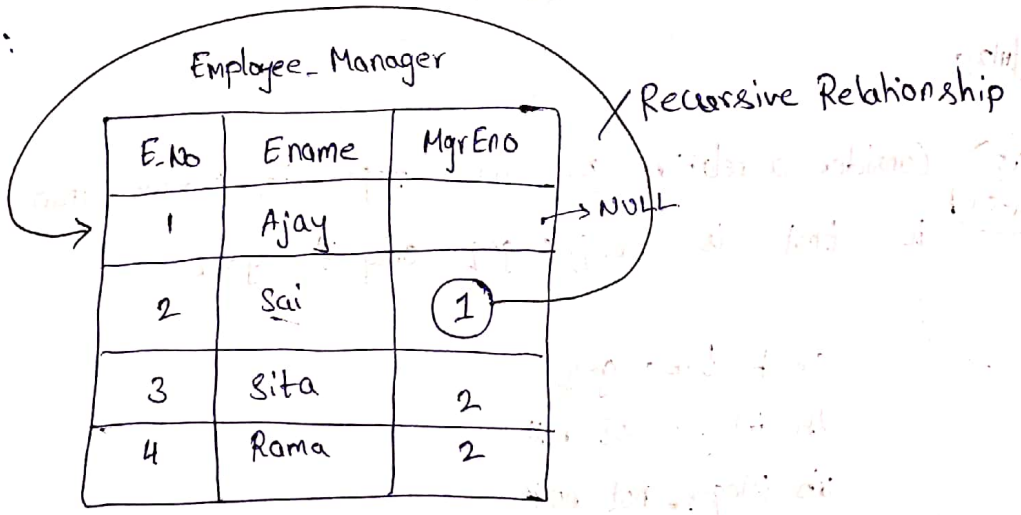
set default

→ Same as above condition but a default value is given rather than a null value.

But if this default value is not present under Pk column of parent table, a violation will occur.

→ Fk may also refer to Pk of the same relation.

Eg:



Q1 / 4-05 Consider below relation R where A is ~~foreign~~ Pk and C is Fk refering A with on delete cascade.

If <2,4> is delete then what are the extra tuples deleted to preserve the referential integrity:

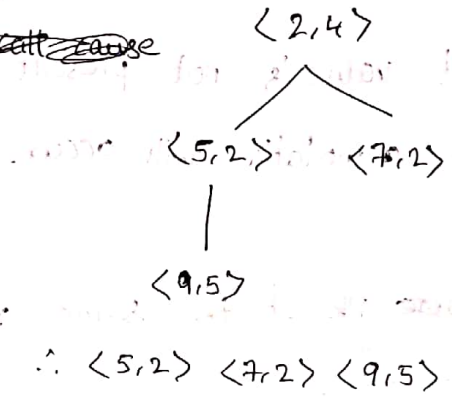
a) (5,2)
b) (3,4)
c) (5,2) (7,2) (9,5)
d) (3,4) (4,3) (6,4)

R

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

Sol:

del <2,4>
will ~~cause~~



∴ <5,2> <7,2> <9,5>

10/11/20

Q2 / a-01 Consider a relation geq which represents "greater than or equal to" that is, $(x,y) \in geq$ only if $y \geq x$

```
{
    create table geq
    lb integer not null
    ub integer not null
    primary key lb
    (foreign key (ub) references geq on delete cascade)
}
```

which of the following is possible if a tuple (x,y) is deleted?

a) A tuple (z,w) with $z > y$ is deleted
b) A tuple (z,w) with $z > x$ is deleted
c) A tuple (z,w) with $w < x$ is deleted
d) The deletion of (x,y) is prohibited.

on dd cascade

Sol: <lb, ub>

Consider ⑧ <8,10> is to be deleted

then every tuple of form <a,8> is deleted

then every tuple of form <b,a> is deleted

and so one

Here $b \le a \le 8 \le 10$

$\therefore b < 10$

<z,w> is deleted

$\Rightarrow z \le w$ & $w \le x$

$\therefore$ opt ©

Here w<x is correct but not w ≤ x

Eg: <2,3>
<1,8>

deletion of <2,3> doesn't cause deletion of <1,3>

Note:

In on delete cascade, ~~on delete upda~~ on update cascade, on delete set null, on delete set default, the child table tuple are modified first.

P/13 If the question is like

s is foreign key referencing p in table T_1 with on delete cascade and on update cascade then

<3,8> will try to delete <8,3>
(T_1) (T_2)

now <8,3> (T_2) will try to delete <3,8>, <5,8>, <9,8> of T_1

so this infinite loop continues.

Conceptual Database Design using E-R model:

→ ER model : Entity Relationship model

→ we converst concept (requiments) to E-R model.

E-R Components :

(i) Entity: Read world object. (In general entities are nouns)

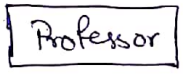
Eg : Professor, course

(ii) Entity set :

Set of entities

Eg : set of professors.

→ Represented as rectangle



(iii) Relationship:

It is association blw two ~~entities~~. or more entities.

Eg : Professor teaches a course.

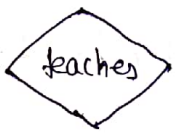
↓
relationship

(Generally relationship is identified by verbs)

(iv) Relationship set :

set of similar relationships.

represented using rhombus

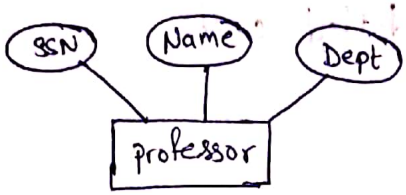


(v) Attribute:

Every entity is associated with set of attributes.

It is represented using oval.

Eg:



Classification of Attributes:

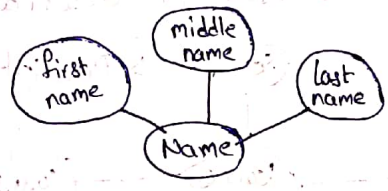
(i) Simple Attribute:

An attribute is said to be simple, if it cannot be divided into further attributes.

(ii) Composite Attribute:

It is can be divided into set of simple attributes.

Eg:



Based on no of values associated:

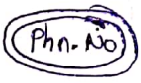
(i) Single valued attribute:

The attribute can take maximum of one value only.

(ii) Multivalued attribute:

The attribute can take more that one value.

It is represented using double ellipse.



Based how the attributes value is stored:

(i) Stored attribute:

An attribute which is stored in database and its value cannot be ~~whose value is not going to be changed~~ derived from other attributes.

Derived attribute.

An attribute is said to be derived if can be derived from ~~other attribute~~ a set of attributes.

It is represented using dotted oval.

Eg → (Age)

key attribute:

It is the attribute which identifies every entity in the entity set.

key attribute is represented as underlined attribute.

(Roll.No)

Descriptive attribute

Descriptive attribute is used to describe a relation.

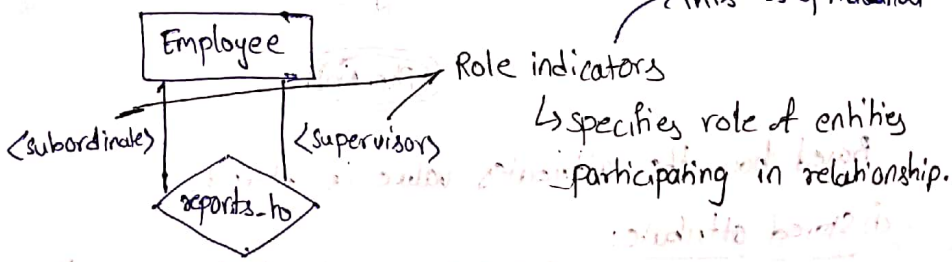
Eg:

Relationship Degree:

Degree of a relationship is no of entities associated with a relationship.

(i) Unary relationship / Recursive Relationship

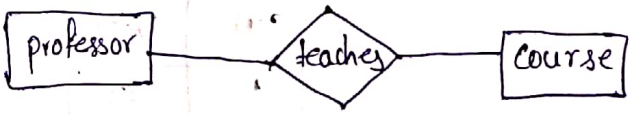
Relationship with degree 1.



This is also known as recursive relationship.

(ii) Binary relationship:

Relationship of degree 2.



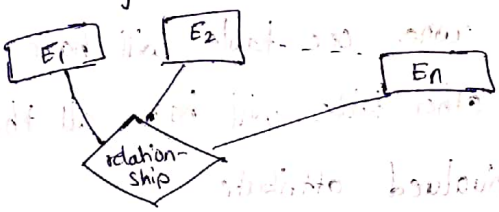
(iii) Ternary relationship:

Relationship of degree 3.

a ~~Similar~~

n-ary relationship:

Relationship of degree n.

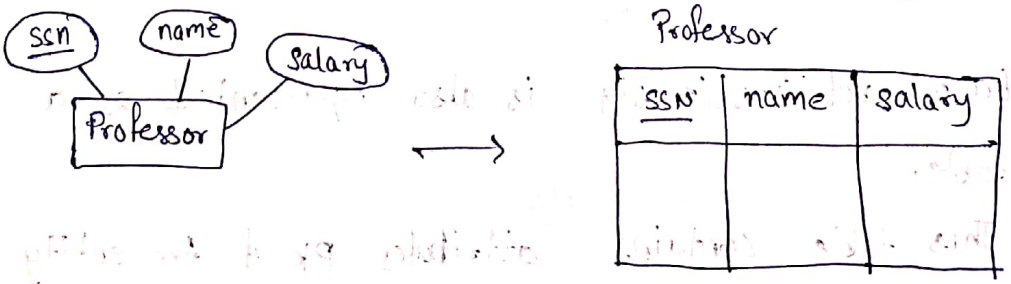


11/11/20

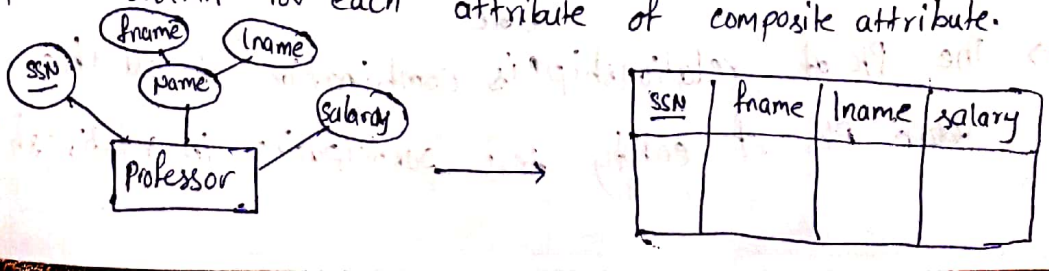
ER to relational model

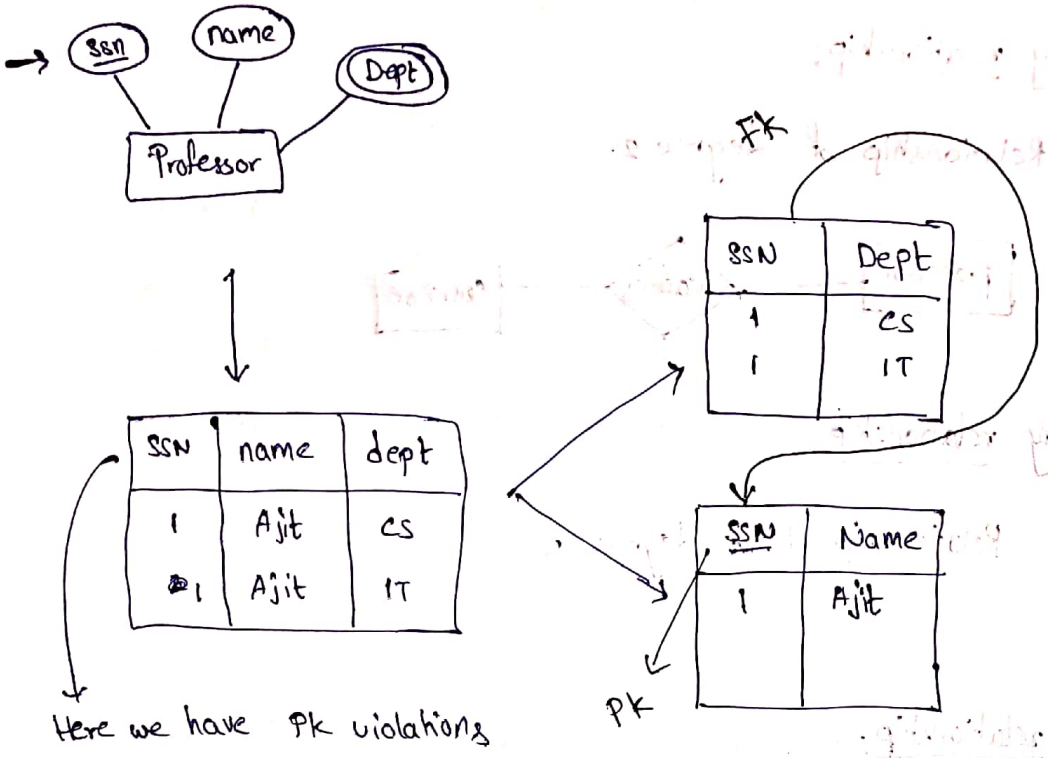
→ Every entity set is represented as one table.

→ The attributes of entity set becomes attributes of table.



→ If we have composite attribute in ER model, then we create separate column for each attribute of composite attribute.





So the table is divided into two tables where one table will have Pk and multivalued attribute and other table will have all the attributes other than the multivalued attribute

→ If an entity set has multivalued attribute then it has to be represented using 2 tables.

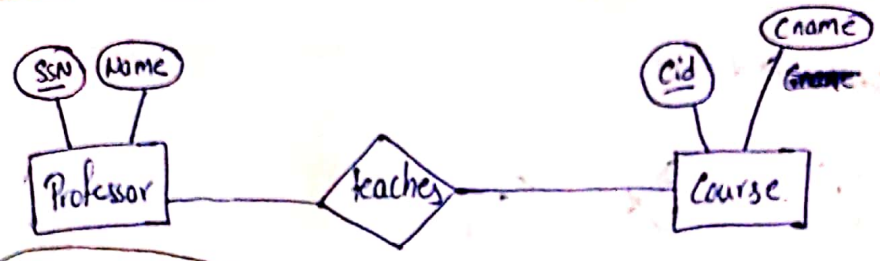
Converting relationship set into relational model

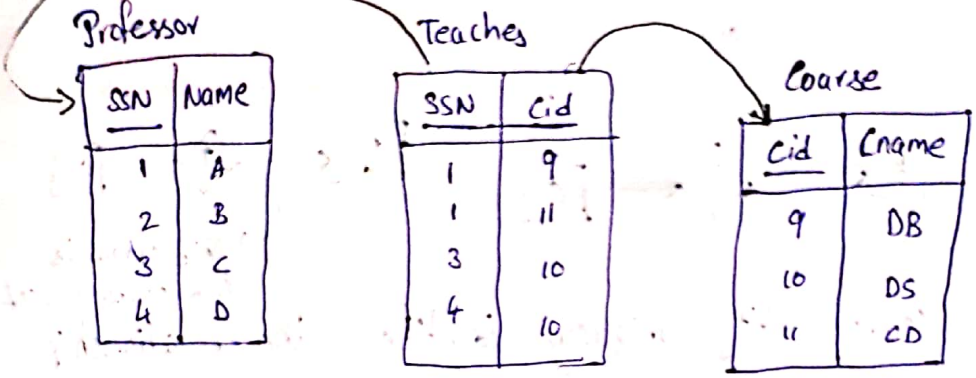
→ Each relationship set is also represented as a table.

→ This table contains attributes Pk of the entity sets participating in relationship as Fk, and other attributes specific to the relationship take table

→ The Pk of relationship is combination of all the Pks of entity sets participating in relationship.

Eg:





SSN → Fk

Cid → Fk

{SSN, Cid} → Pk

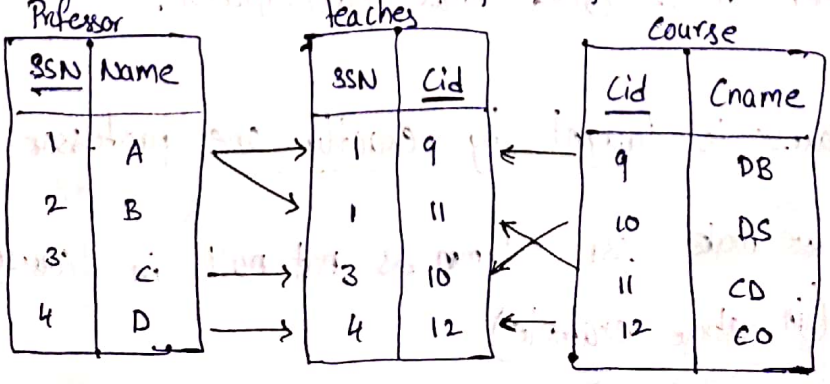
key Constraints in E-R model:

If for each entity in the entity set there is atmost one relationship in the relationship set then the entity set is said to be a key constraint.

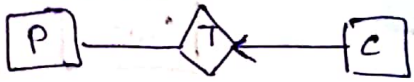
The key constraint is represented with arrow (from entity set to relationship set)

Eg: Each course is taught by atmost one professor.

Here course is the key entity set.



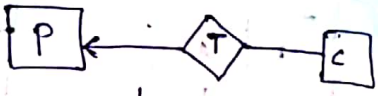
It is represented as



But sometimes combining may introduce NULL values which will lead to wastage of space.

But combining can make certain operations (involving both tables) execute faster.

(or)



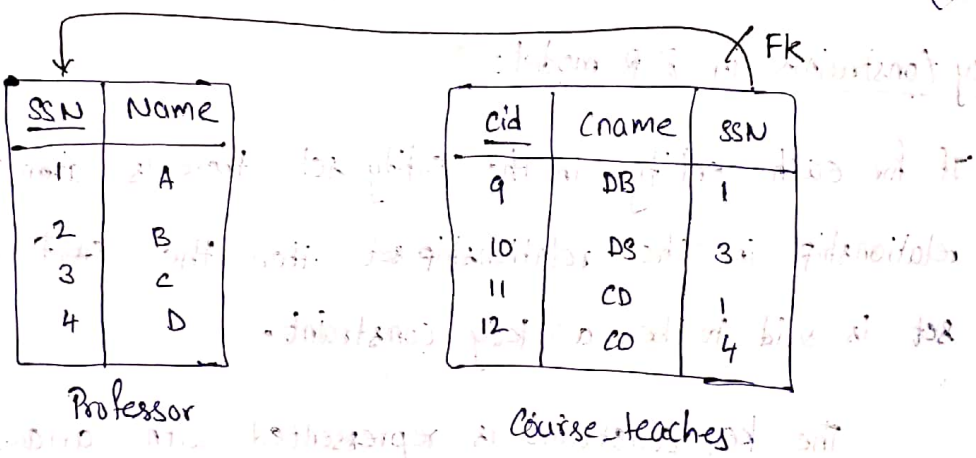
This arrow is from course to prof

→ Here each course entity uniquely identifies the relationship.

→ Thus the key of relationship table is cid.

→ Thus we combine teaches & course tables.

→ The resultant tables are:



Participation Constraints:

total participation:

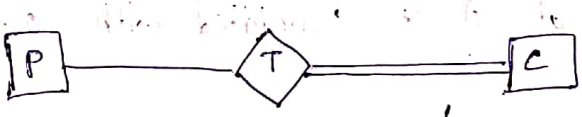
If every entity of an entity set participates in the relationship then it is called total participation.

Eg: Each course is taught by exactly one professor

So Here we make ssn column as not null in course_teaches table (in above example)

It is represented using ~~double line~~ ~~directed from entity~~ ~~set to the relationship~~

↳ total participation with key constraint



↳ total participation.

Partial Participation:

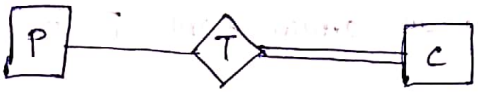
If ~~some~~ some entities in the entity set are not participating in a relationship then it is called partial participation. It is represented using ——

Note:

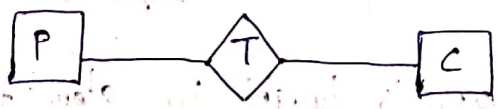
atmost one :
(every course taught by atmost one prof)

exactly one :
(every course taught by exactly one prof)

atleast one :
(every course taught by atleast one prof)



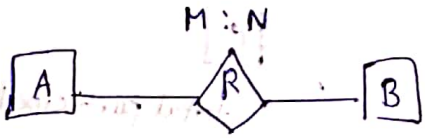
0 (or) many :
(every course is taught by 0 (or) more prof)



Course entity is not a key in this case.

Cardinality ratios:

(i) M:N (many to many)



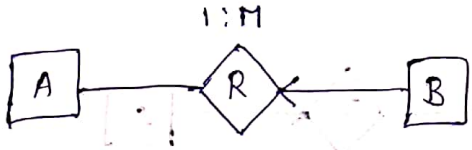
→ Each entity of A is associated with 0 (or) more entities of B.

→ Each entity of B is associated with 0 (or) more entities of A.

→ To represent this in relational model, we need three tables: ~~A,B,R~~ A, R, B

Pk of R is {Pk of A, Pk of B}

(ii) 1:M (one to many)



→ Each entity of A is associated with 0 (or) more entities of B.

→ Each entity of B is associated with 0 (or) 1 entity of A.

So we draw an arrow from B to R.

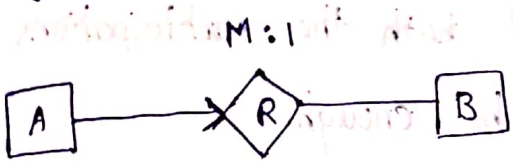
→ It is similar to the previous example of each course is taught by atmost one professor

→ This represented using 2 tables.

↳ (A & R_B)

Pk of R_B is → Pk of B

(iii) M:1 (Many to one)



→ It requires 2 tables (A-R & B)

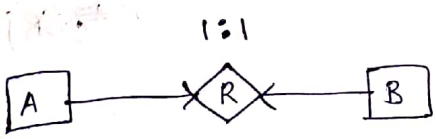
Pk of A-R is Pk of A

A-R has Pk of B as FK

Note:

In 1:M & M:1 '→' is from 'M' side.

(iv) 1:1



⇒ Each entity in A is associated with atmost one entity of B.

→ Each entity in B is associated with atmost one entity of A.

→ It is represented using 2 tables.

either (AR, B) or (A, RB)

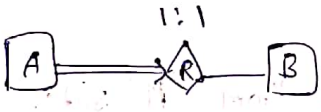
Pk = Pk of A
FK = Pk of B

→ we cannot represent this using one table because for some key value of A, corresponding B can be null and vice versa. In this case we cannot choose primary key.

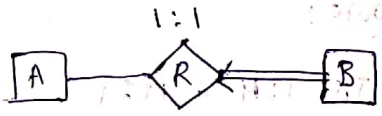
Note:

→ In 1:1 cardinality, if both the participations are total then one table will be enough.

→ In 1:1 cardinality, if total participation is only from one side then we need minimum one table



pk: pk of B



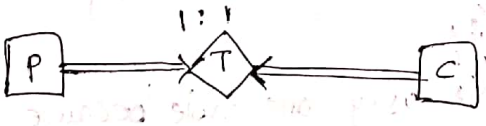
pk: pk of A

P/5 arrow is present from car to purchase.

∴ car is key in the relationship.

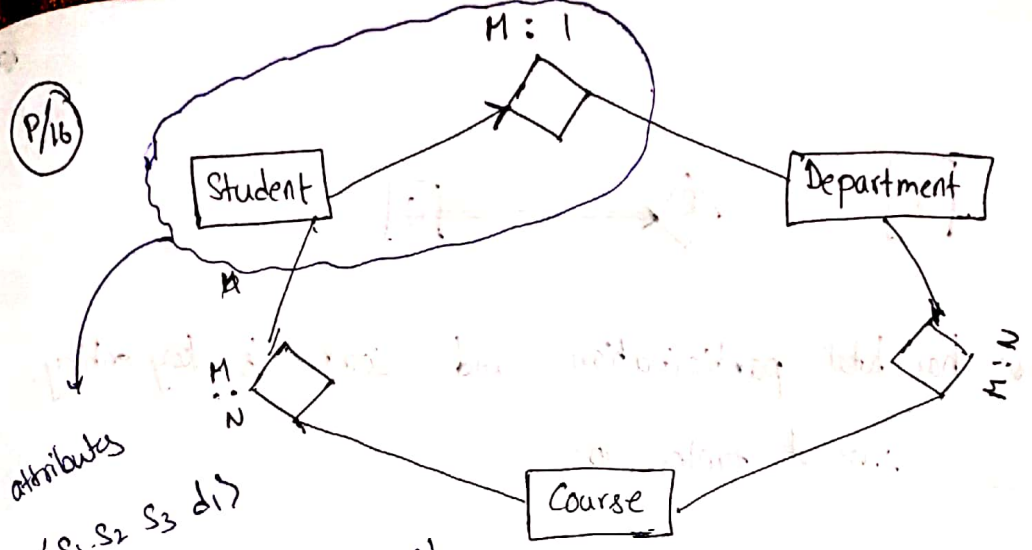
∴ Each car has atmost one dealership

P/15



∴ 1 table

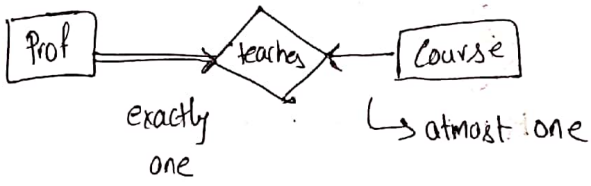
P/16



d_1 is key of department

(In ER model key is always a single attribute. So we added only d_1)

P/17



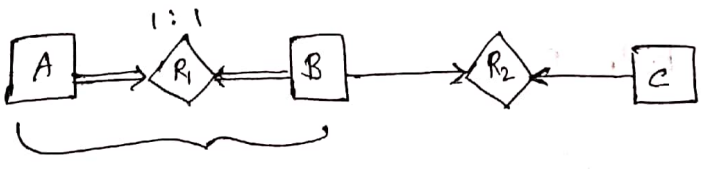
SSN	Name	cid	Cname
1	A	9	DB
null	null	10	CO

cid column can never have null

∴ cid is pk

12/11/20

P/18



can be represented using one table

Now R_2 can be merged with AB or C

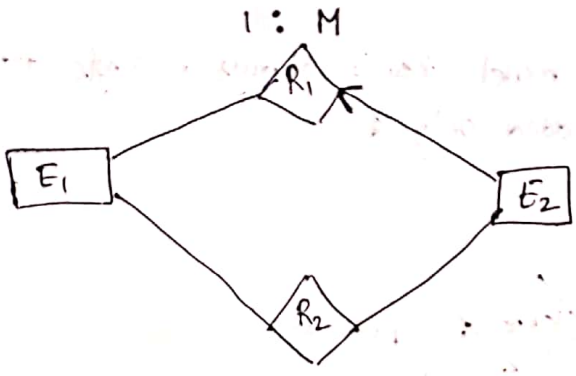
∴ 2 tables.

P/19

Course has total participation and course is key entity.

∴ no of tuples = 100

P/20



- ~~R1 & E2 can be com~~

Initially we start with 4 tables

E_1, E_2, R_1, R_2

we can combine E_2 & R_1

∴ we finally have 3 tables

E_1, E_2R_1, R_2

P/21

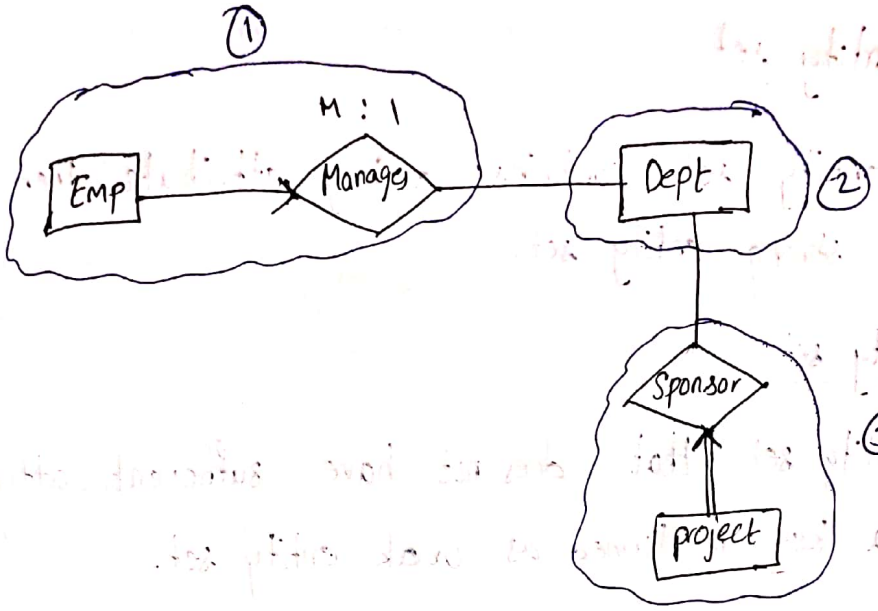
E_2 has a multivalued attribute.

∴ It require 2 tables.

we can combine E_2 & R.

So we have 3 tables.

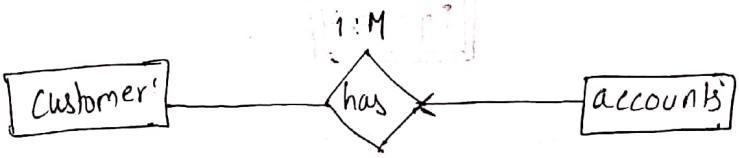
P/22



∴ 3 relations

P/23

a)



2 tables

b) M : N

3 tables

c) 1 : M

2 tables

d)

∴ 1 table

Strong entity set:

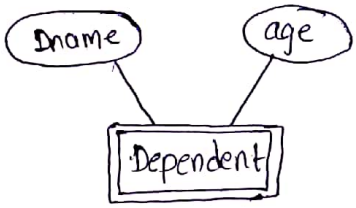
If an entity set contains a key attribute. then it is called strong entity set.

Weak entity set:

An entity set that does not have sufficient attributes to form a key is termed as weak entity set.

weak entity set is represented using double rectangle.

Eg:



→ • A weak entity set participates in a relationship and it is identified using key of another entity set (strong entity set).

This strong entity set is called owner entity.

This relationship is called weak relationship / identifying relationship

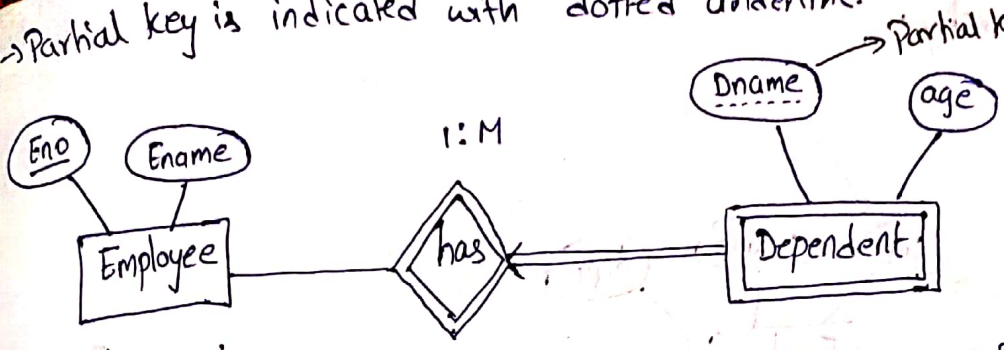
→ The cardinarity of relationship from owner to weak is always 1:M

→ weak entity must have total participation.

→ The key of weak entity set is {key of owner + partial key}

→ The attribute of weak entity set which when combined with key of owner behaves as key of weak entity set is called partial key.

→ Partial key is indicated with dotted underline.



ENO	Ename
1	P
2	Q
3	R
4	S
5	T

Eno	Dname
1	A
1	B
2	A
2	B
3	C
4	C

partial key

Eno	Dname	age
	A	12
	B	6
	A	10
	B	9
	C	3
	C	3

Foriegn key

Eno	Dname	age
1	A	12
1	B	6
2	A	10
2	B	9
3	C	3
4	C	3

key

{Eno, Dname}

↓ key of owner

↓ partial key

Existance Dependency

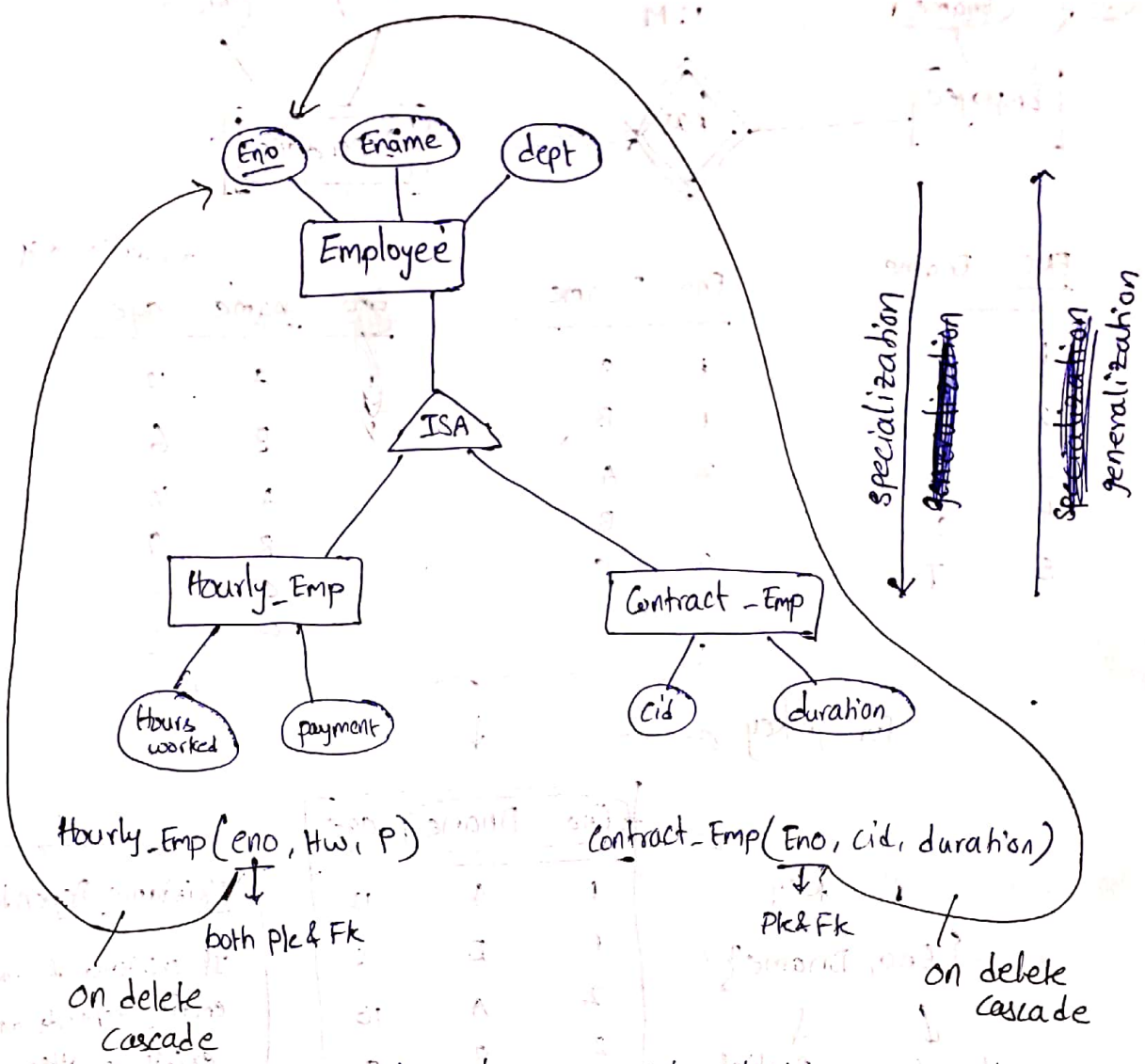
If existence of one enitity depends on exists of other entity, then it is called existence dependency

→ Existence of a weak entity is possible & meaningful only when its owner entity exists.

∴ The Fk key 'Eno' in weak entity set relation has on delete cascade Fk constraint.

∴ Fk of weak enitity set always has on delete cascade relationship.

Class Hierarchy:



→ whenever an entity of employee is deleted the corresponding entity must be deleted from hourly_emp or contract_emp.

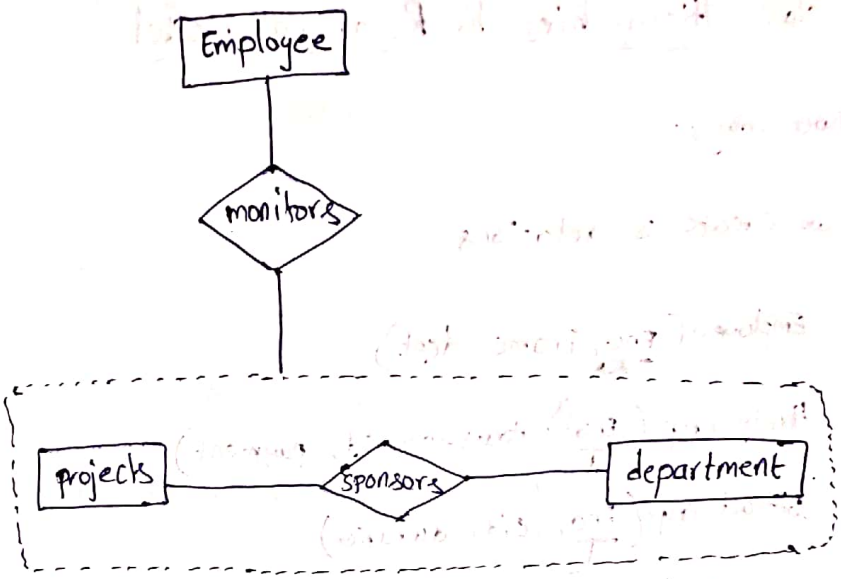
∴ Fk constraint is on delete cascade

Aggregation:

Aggregation is relationship over a relationship.

Eg: Consider

Employee monitors the sponsorship of projects by the departmet.



→ This represented with 5 tables.

In ISA hierarchies, we have 2 constraints
(i) overlap: says whether two subclasses can have same entity.
(ii) Covers: says whether the all subclasses must form all the entities of generalized entity set.
→ If overlapping is allowed it has to be specified explicitly
→ covering constraints also has to be explicitly specified

P/24)

3 tables

(i) MR_1
(ii) P
(iii) R_2N

P/25)

$MR_1 \rightarrow \{M_1, M_2, M_3, P_1\}$

$P \rightarrow \{P_1, P_2\}$

$R_2N \rightarrow \{P_1, N_1, N_2\}$

∴ opt (a)

Schema Refinement:

Problems due to redundancy:

(i) Repeated storage

(ii). Insertion problem.: It may not be possible to store certain information unless some other unrelated info is stored as well.

(iii) Updation problem: If one copy of repeated data is update, inconsistency is created unless all the copies are updated

(iv) Deletion problem: It may not be possible to delete certain information without losing other unrelated information.

(i) Repeated storage

Assume we have a requirement that

"Empolyees of same seniority should have same salary".

Eno	Ename	Seniority	salary
1	Ajith	5	50k
2	Raju	5	50k
3	kamal	5	50k
4	Harsha	10	1L
5	sharma	10	1L

(ii) Insertion problem:

If we need to insert

<6 shastry 10 1.2L>

this causes violation to customers requirement.

(iii) Updation problem:

Assume we need to change salary of ajith to 60k.

This causes violation.

(iv) Delehion problem:

If tuples 4&5 are deleted, the association (10-1L) will be lost.

→ The problem's solution is decomposition.

we form below two tables.

Emp:

Eno	Ename	seniority
1	Ajith	5
2	Raju	5
3	kamal	5
4	Harsha	10
5	sharma	10

FK

Emp_salary:

Seniority	Salary
5	50k
10	1L

Now all the 4 problems are solved.

→ Here we say salary is functionally dependent on seniority.

Functional Dependencies

If A & B are attributes of a relationship and if for each value of A there exactly one value of B then we say "B is functionally dependent on A" or "A functionally determines B".

It is denoted as $A \rightarrow B$

If $A \rightarrow B$ then

A	B
x_1	y_1
x_1	y_1
x_2	y_1
x_2	y_1
x_3	y_2
x_3	y_2

valid

A	B
x_1	y_1
x_1	y_2

invalid

Eg: Consider below relation

A	B	C
a_1	b_1	c_1
a_2	b_1	c_1
a_1	b_2	c_1

which of the following FD's are satisfying

a) $A \rightarrow B$ ✗

b) $B \rightarrow C$ ✓

c) $C \rightarrow A$ ✗

d) $C \rightarrow B$ ✗

e) $A \rightarrow C$ ✓

f) $B \rightarrow A$ ✗

g) $AB \rightarrow C$ ✓

h) $AC \rightarrow B$ ✗

i) $BC \rightarrow A$ ✗

j) $AC \rightarrow C$ ✓

Trivial FD :

The FD in which set of RHS attributes is subset to LHS attributes.

otherwise it is called non-trivial FD.

13/11/20

Properties of Functional Dependencies:

(i) Reflexivity:

If $Y \subseteq X$, then

$X \rightarrow Y$ is an FD

Eg: $AB \rightarrow B$

The FDs under this property are trivial FDs

(ii) Augumentation:

If $X \rightarrow Y$ then $XZ \rightarrow YZ$ is a valid FD

(iii) Transitivity:

If $X \to Y$ & $Y \to Z$ are two valid FDs then

$X \to Z$ is also a valid FD

(iv) Union:

If $X \to Y$ and $X \to Z$ then $X \to YZ$

(v) Decomposition:

If $X \to YZ$ then

$X \to Y$ and $X \to Z$

Closure set of FD's:

If F is a set of function dependencies,

closure of F, (F^+) is set all valid FDs possible from given set of dependencies.

Eg: Consider relation R(ABC), and its FDs $F: \{A \to B, B \to C\}$

Computing F^+

$A \to B, B \to C$

$A \to C$ (transitivity)

$A \to BC$ (union)

$AC \to BC$ (Augumentation)

$AC \to B$; $AC \to C$ (Decomposition)

⋮

Closure set of attributes:

If X is a set of attributes, then closure of X, (X^+) is set of all attributes that can be determined by X.

Eg: Consider

R(ABC) $F: \{A \to B, B \to C\}$

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

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

$C^+ = \{C\}$

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

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

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

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

→ If ~~cot~~ closure of X includes all the attributes of relation, then X is a key

∴ In above example A, AB, AC, ABC are keys

and A is a candidate key.

AB, AC, ABC are super keys but not ck.

no of cks: 1

no of sks: 4

Eg: Consider R(ABCDE)

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

Find

(i) AB^+

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

$\because AB \to C$, $B \to D$

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

(ii) B^+

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

(iii) E^+.

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

(iv) BE^+

$E \to A$

$AB \to C$

$B \to D$

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

Applications of attributes closures:

(i) Additional FDs can be determined:

Given a set of dependencies and if it is asked to check if $X \to A$ is possible is not.

If $A \subseteq X^+$ then we say $X \to A$ is possible.

Q3 / G-05 Consider R(ABCDE) the set of FDS:

$A \to B$; $A \to C$; $CD \to E$; $B \to D$; $E \to A$;

which of the following FDs is not implied

a) $CD \to AC$

b) $BD \to CD$

c) $BC \to CD$

d) $AC \to BC$

Sol:

a) $CD \to AC$

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

b) $BD \to CD$

$(BD)^+ = \{B, D\}$

$\therefore$ false

c) $BC \to CD$

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

d) $AC \to BC$

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

(ii) Determining equivalence of two FD sets:

Given two sets of FDs, say F & G

F & G are said be equivalent iff $F^+ = G^+$

→ But finding equivalence in this process is a tedious.

→ So we find if F covers G and G cover F.

If F covers G and G covers F, then F & G are said to be equivalent.

→ we say F cover G,

if all the dependencies of G can be derived from F.

$\therefore \{(F \text{ cover } G) \wedge (G \text{ covers } F)\} \Leftrightarrow F \cong G$

Eg: Consider

$F: \{A \to B, AB \to C, D \to AC\}$

$G: \{A \to BC, D \to A\}$

Checking F covers G:

$A \to B$

$AB \to C$

(i) $A \to BC$

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

(ii) $D \to A$

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

$\therefore$ F covers G.

Cheking G covers F

(i) $A \to B$

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

(ii) $AB \to C$

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

(iii) $D \to A$

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

$\therefore$ G covers F $\quad \therefore F \cong G$

(Q4) $F: \{A \to C, AC \to D, E \to AD, E \to H\}$

$G: \{A \to CD, E \to AH\}$

∴ equivalent

(iii) Finding minimal set of FDs / canonical form:

A set of FDs F is minimal if it satisfies the below conditions:

(i) Every dependency in F has a single attribute on its right side.

(ii) we cannot replace any $X \to A$ in F with $Y \to A$, where Y is a proper subset of X and still have equivalent set to F.

(iii) No dependency can be removed such that the set is equivalent to F.

Procedure to find canonical cover:

(i) Minimize LHS of each FD:

Eg: Consider $AB \to C$

'A' can be deleted if B^+ contains A

Eg: $F: \{AB \to C, B \to A\}$

Canonical cover, $G: \{B \to C, B \to A\}$

A set may even have more than one minimal sets possible

Eg: Consider

$$F: \{AB \to C, B \to A, A \to B\}$$

A^+ contains B

B^+ contains A

$G_1: \{A \to C, B \to A, A \to B\}$ $\quad$ $G_2: \{B \to C, B \to A, A \to B\}$

Here $F \cong G_1 \cong G_2$

(ii) Minimize RHS of each FD:

First make sure that right hand side has only one attribute (using decomposition)

Let $F: \{A \to B, B \to C, A \to C\}$

$A \to B$

Cannot be obtained from $B \to C$ & $A \to C$

$B \to C$

Cannot be obtained from $A \to B$ & $A \to C$

$A \to C$

Can be obtained from $A \to B$ & $B \to C$

$\therefore A \to C$ is redundant.

Thus F can be reduced to

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

→ Thus we find canonical cover performing above two step.

(Q5) $F: \{AB \to C, A \to B, D \to AC\}$

Find minimal set

Sol:

Step 1: Minimize LHS

i) $AB \to C$

$A^+ = \{A, B, C\} \Rightarrow$ B can be deleted

$B^+ = \{B\} \Rightarrow$ A can't be deleted

A^+ contains B

So remove B

$A \to C$

$\therefore F: \{A \to C, A \to B, D \to AC\}$

Step 2:

rewrite such that RHS has one attribute

$A \to C$

$A \to B$

$D \to A$

$D \to C$

- Check if $A \to C$ is derivable from other dependencies

i.e.,

Compute A^+ from $A \to B, D \to A, D \to C$

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

$\therefore A \to C$ can't be removed

- $A \to B$

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

$A \to B$ can't be removed

- $D \to A$

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

$\therefore D \to A$ can't be removed

- $D \to C$

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

$\therefore D \to C$ can be removed

~~Final~~ $\therefore$ Final minimized set is

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

$\therefore F: \{A \to BC, D \to A\}$ is

minimal set / canonical set / irreducible set.

(Q2) $F: \{AB \to C, A \to B, B \to C\}$. Find minimal set

Soln:

Step 1:

~~A^+ can't~~ $A^+ = \{A, B, C\}$

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

$\therefore F: \{A \to C, A \to B, B \to C\}$

Step 2:

$A \to C$ can be obtained from $A \to B$ & $B \to C$

$\therefore F: \{A \to B, B \to C\}$

(P/15) $V \to W$

$VW \to X$

$Y \to VX$

$Y \to Z$

Step (i)

$VW \to X$

~~$V^+ = \{W\}$~~ V^+ contains W

$W^+ = \{W\}$

$\therefore$ remove W

$V \to W;\ V \to X;\ Y \to VX;\ Y \to Z$

Step (ii)

$V \to W$
$V \to X$
$Y \to V$
$Y \to X$
$Y \to Z$

$Y \to X$ can be removed

$\because Y \to V$ & $V \to X$

$\therefore$ reduced set is

$V \to W$
$V \to X$
$Y \to V$
$Y \to Z$

P/11

$\{A \to B, AB \to C, A \to BC, B \to C\}$

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

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

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

P/12

$\{A \to B, CD \to A, CB \to D, CE \to D, AE \to F, AC \to D\}$

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

$D^+ = \{D\}$

Step (i) :- ~~$\{A \to B, CD \to A, CB \to D, CE \to D, AE \to F, AC \to D\}$~~

Step 1 : No minimization can be done in step(i)

Step 2 :

$AC \rightarrow D$ can be removed

$\therefore (AC)^+ = \{A, B, C, D\}$ (from remaining 5 dependencies)

$\therefore 5$

14/11/20

P/13

$BCD \rightarrow A$

$C \rightarrow E$

$E \rightarrow B$

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

$\therefore$ B can be eliminated from $BCD \rightarrow A$

$\therefore$ $CD \rightarrow A$

$C \rightarrow E$

$E \rightarrow B$

P/14

$BC \rightarrow A$

$EF \rightarrow A$

$B \rightarrow D$

$D \rightarrow EF$

$C \rightarrow F$

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

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

$E^+ = \{E\}$

$F^+ = \{F\}$

$\therefore$ no minimization on left size

$BC \rightarrow A$:

$(BC)^+ = \{B, C, D, E, F, A\}$ $\therefore$ $BC \rightarrow A$ can be removed

P/16

$A^+ = \{A, B, C, G, D\}$ B can be removed from $ABE \rightarrow CDGH$

∴ ~~no minimization left side~~.

$A \rightarrow B$ ——— essential
$A \rightarrow C$ ——— essential
$ABE \rightarrow C$ ——— not needed
$ABE \rightarrow D$ ——— not needed
$ABE \rightarrow G$ ——— not needed.
$ABE \rightarrow H$ ——— needed
$C \rightarrow G$ ——— not needed
$C \rightarrow D$ ——— needed
$D \rightarrow G$ ——— needed
$E \rightarrow F$ ——— needed

∴ $\{A \rightarrow B; A \rightarrow C; ABE \rightarrow H; C \rightarrow D; D \rightarrow G; E \rightarrow F\}$

(iv) Finding the keys of a relation:

If X is set attributes and if X^+ contains all the attributes then X is called super key.

If X is minimal then we call it candidate key.

Eg: Consider

R(ABC) $F: \{A \rightarrow B, B \rightarrow C\}$

Find closure of LHs of all dependencies

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

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

now 'A' is ck

Ck: A

Sk: A, AB, AC, ABC

∴4 super keys

Eg: Find Cks and Sks

R(ABCD) ∴ F: {AB → C, A → D}

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

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

∴ AB is ck.

Ck : AB

Sk : AB, ABC, ABD, ABCD

Eg:

R(ABCD)

F: {A → B, AB → C, B → D}

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

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

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

Ck : A

Sk : A, AB, AC, AD, ABC, ABD, ACD, ABCD

(8 combinations of B, C, D are possible)

Eg:

R(ABCD)

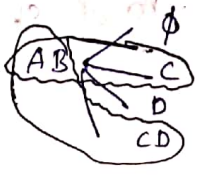
F: {AB → C, BC → AD}

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

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

Ck : AB, BC

Sk :



∴ no of Sks = 4 + 4 − 2 = 6

In these kind of situation we compute no of sks by combining (union two cks) and find no of possibilities and subtract it

AB → 4 possibilities BC → 4 possibility

ABC → 2 possibility

$\therefore 4+4-2=6$

Eg: R(ABC) F: { A → BC , BC → A }

$A^+ = [A, B, C]$

$(BC)^+ = [A, B, C]$

Ck : A, BC

Sks :

A → 4 ways

BC → 2 ways (−) ABC → 1 ways

$\therefore (4+2)-1 = 5$

Eg: R(ABC)

F: [A → B, B → C, C → A]

Ck : A, B, C

Sk : since all attributes are key

every combination of attributes is sk.

$\therefore$ no of sks = $2^3 - 1 = 7$

Eg: R(ABCD)

F: {A → B, C → D}

Find Cks.

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

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

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

∴ AC is a candidate key

Eg: R(ABCD)

F: {A → BC, B → C}

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

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

$D^+ = \{D\}$

∴ AD is Ck

Eg: R(ABC)

F: {AB → C, C → A}

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

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

Here B is independent attribute

i.e., no attribute determines B

∴ Ck: AB, BC

Eg: R(ABCD)

F: ~~{AB→C, B→D, D→A, D→C}~~

F: {AB→C, A→D, D→A, C→D}

Here B is independent attribute

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

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

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

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

AB is one Ck.

A, B are prime attributes.

To obtain other Cks check if there is any dependency such that prime attribute on RHS. If any is found, then replace it with its LHS in ck.

Ck → AB

In D→A, A is present on RHS

∴ DB is also Ck

In C→D, D is present on RHS

∴ CB is Ck

∴ Cks : AB, BC, BD

Finding Sks:

AB → 4 BC → 4 BD → 4

ABC → 2 ABD → 2 BCD → 2

ABCD → 1

∴ (4+4+4) − (2+2+2) + 1 = 12 − 6 + 1 = 7

Eg: R(ABCDE)

F: {AB → C, BD → E, DE → B}

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

$(BD)^+ = \{B, D, E\}$

$(DE)^+ = \{B, D, E\}$

A, D are independent attributes.

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

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

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

Cks: ABD, ADE

∴ A, B, D, E are prime attributes.

∴ no of Cks = 2

Eg: R(ABCDE)

F: {A → B, BC → D, DE → A}

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

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

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

Here C, E are independent attributes.

$(CE)^+ = \{C, E\}$

* Independent attributes must exist in Ck.

So we try adding different combinations to CE

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

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

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

∴ 3 CKs : ACE, BCE, DCE

P/24

R(ABCDEFGH)

$(CH)^+ = \{C, H, G\}$

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

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

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

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

Here D is independent attribute & $D^+ = \{D\}$

AD, BD, ED, FD are CKs

∴ 4 CKs

P/19

~~$C^+ = \{C E F\}$~~

Here C, E are independent attributes.

CE must be present in key

∴ opt (a)

P/23

E, H are independent attributes and must be part of key

$(EH)^+ = \{E, H, C\}$

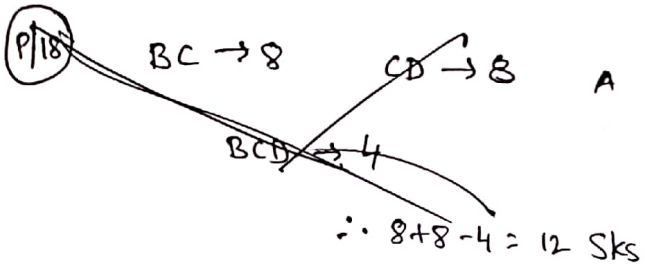
from options opt (d)

P/17 $AB \rightarrow 2^4 = 16$ $AE \rightarrow 2^4 = 16$

$ABE \rightarrow 2^3 = 8$

$\therefore 16+16-8 = 24$

P/18



$\therefore 8+8-4 = 12$ Sks

P/18 $AB \rightarrow 8$ $BC \rightarrow 8$ $CD \rightarrow 8$

$ABC \rightarrow 4$ $BCD \rightarrow 4$ $ABCD \rightarrow 2$

$ABCD \rightarrow 2$

$(8+8+8) - (4+4+2) + 2 = 16$

P/20 A, B, C, D, k are independent attributes

$\therefore$ opt (d)

P/21 A, B are independent

~~opt (b)~~ b) $(AB)^+ = \{A, B, C\}$

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

$\therefore$ opt (d)

P/22 H is independent

$\therefore$ opt (b)

P/25

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

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

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

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

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

AB, F are CKs

A, B, F are prime attributes

In $C \rightarrow A$,

A is on RHS

∴ In AB, we can replace A with C

∴ CB is CK

A, B, C, F are prime attributes

∴ 3 CKs (AB, CB, F)

P/26

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

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

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

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

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

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

AB is CK

A, B are prime attributes

AD is CK

BE is CK

now A, B, D, E are prime attributes

∴ BC is key

∴ ~~BA is key~~

AB, AD, BC, BE

replace A with E

EB, ED, AB, AD, BC,

↓

CD

P/27

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

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

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

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

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

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

∴ Ck ⟹ D, ABH

$A \rightarrow B$

∴ ABH is not ck

AH is ck

B, D, AH

∴ 2

Properties of Decomposition:

(i) Lossless join decomposition

(ii) dependency preserving decomposition

(i) Lossless join decomposition:

The decomposition of R into R_1 & R_2 is said to be lossless iff the join of R_1 & R_2 gives R.

Otherwise it is called lossy decomposition.

→ When R is divided in R_1 & R_2, for the decomposition to be lossless there must be a common attribute between R_1 & R_2.

The common attribute must be either key of R_1 or key of R_2.

This common attribute if it is key of R_1, then it acts as Fk in R_2.

Now we say parent child relationship is established blw R_1 and R_2.

The decomposition is lossless iff parent child relationship is maintained properly.

Eg: Consider

$R(ABC)$ $F: \{A \rightarrow B\}$

is decompose into $R_1(AB)$ $R_2(BC)$.

Check if this decomposition is lossy or lossless.

$R_1 \cap R_2 = B$

The common attribute has be key of R_1 or key of R_2.

~~None But it is~~ ,

But B is key for neither of the tables. ∴

∴ It is lossy decomposition.

Eg: $R(ABC)$ $F: \{A \to B\}$

$R_1(AB)$ $R_2(AC)$

$R_1 \cap R_2 = \{A\}$

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

A determines attributes of R_1

∴ R_1 is parent

R_2 is child

∴ lossless decomposition.

Eg: $R(ABCDE)$ $F: \{AB \to C, B \to D, D \to E\}$

~~Let R be~~ decomposed into $R_1(ABC)$ $R_2(BD)$ $R_3(DE)$

Consider joining $R_1 \& R_2$

$R_1 \cap R_2 = B$

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

∴ $R_1 \& R_2$ a parent child relation ship exists

↳ lossless join

$R_1 R_2(ABCD)$ $R_3(DE)$

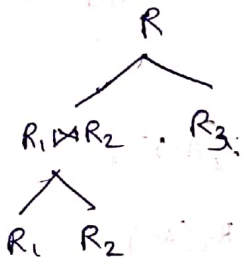
$R_1 R_2 \cap R_3 = D$

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

D can determine all attributes of R_3

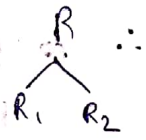
$\therefore R \Leftarrow R_1, R_2, R_3$ is lossless decomposition

The decomposition is



iii) Dependency Preserving decomposition:

A decomposition is said to be dependency preserving iff $F^+ = (F_1 \cup F_2)^+$



F is dependencies of R

F_1 & F_2 are dependencies of R_1 & R_2

Eg: R(ABC) $F: \{A \to B, B \to C, C \to A\}$

is decomposition $R_1(AB)$ $R_2(BC)$ dependency preserving

$R_1(AB)$ – $F_1: \{A \to B\}$

$R_2(BC)$ – $F_2: \{B \to C\}$

~~$C \to A$ in F can't be obtained from F_1 & F_2~~

~~∴ the decomposition is not dependency preserving.~~

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

∴ we can add $B \to A$ to F_1 and

$C \to B$ to F_2

F_1 : $A \to B$, $B \to A$ $\qquad$ F_2 : $B \to C$, $C \to B$

Now we see if $C \to A$ can be obtained from F_1 & F_2

Find c^+ in F_1 & F_2

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

$\Rightarrow C \to A$

∴ The decomposition is dependency preserving

Procedure:

(i) Write all direct dependencies

(ii) Now check for obtain indirect dependencies in F.

we do this by writing all indirect dependencies to

$F_1, F_2, F_3 \ldots$

Now we see if given indirect dependency is possible or not.

Note:

→ A decomposition is said to be good if it is both lossless and dependency preserving.

P/01

a) $R_1(BC)$ $R_2(AD)$

no common attribute

~~lossless~~ ∴ lossy decomposition

$R_1(BC)$ $R_2(AD)$

$F_1: B \to C$ $F_2: D \to A$

∴ Dependency preserving

but it is not a good decomposition

2) lossless

$R_1(ACD)$ $R_2(BC)$

F_1: $C \to A$ F_2:

$C \to D$

$A^+ = \{A\}$

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

$D^+ = \{D\}$

$B^+ = \{B\}$

$C^+ = \{C\}$

we need to check for $AB \to C$

find $(AB)^+$ using F_1 & F_2

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

∴ not dependency preserving

3) lossless decomposition

$R_1(ABC)$ $R_2(AD)$

F_1: $A \to BC$

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

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

$C \to AB$

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

$D^+ = \{D\}$

∴ $A \to D$

Now $F_1 \cup F_2 = \{A \to BC, C \to AB, A \to D\}$

now we need check if $C \to D$ is possible from $F_1 \cup F_2$

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

$\therefore$ dependency preserving

$\therefore$ This is good decomposition

4) lossy decomposition

$R_1(AB)$	$R_2(AD)$	$R_3(CD)$
F_1: $A \to B$	F_2:	F_3: $C \to D$
$B^+ = \{B, C, D\}$	$A^+ = \{A, B, C, D\}$ $D^+ = \{D\}$ $\therefore A \to D$	$C^+ = \{C, D\}$

$\therefore F_1 \cup F_2 \cup F_3 = \{A \to B, A \to D, C \to D\}$

check for $B \to C$

$B^+ = \{B\}$

$\therefore$ not dependency preserving

(P/2)

$R_1(AB)$	$R_2(BC)$	$R_3(BD)$
$A \to B$	$B \to C$ $C \to B$	$B^+ = \{B, C, D\}$ $B \to D$ $D \to B$

$\therefore$ dependency preserving

$\therefore$ ~~dependency preserving~~

$A \to D$ can't be obtained

$\therefore$ ~~not~~ dep. preserving

$R_1 \bowtie R_2$

B is common

$B^+ = [C, D, B]$

$\therefore$ R_1 is child R_2 is parent

$(R_1 \bowtie R_2) \bowtie R_3$

B is common

$B^+ = [C, D, B]$

R_3 is parent

$\therefore$ lossless join

$\therefore$ opt (a)

15/11/20

Normal forms:

1NF:

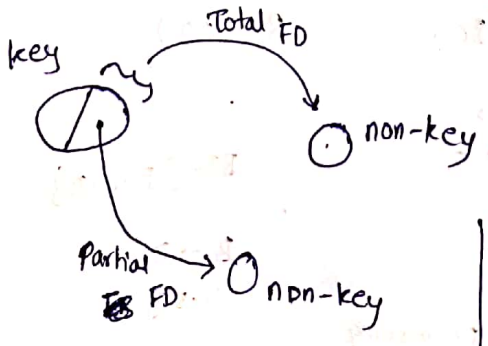
→ 1NF doesn't allow multivalued and composite attributes.

→ 1NF allows only atomic values

→ Every relation is relational model is by default in 1NF.

2NF:

→ 2NF doesn't allow partial FDs. but allows full FD.



PFD: key

part of key → non-key attribute

Eg: If AT in R(ABC)

If A

Eg: Consider R(ABCD) and AB is Ck

AB → C is called full FD

A → D is called partial FD

→ Partial dependencies lead to redundancy

Eg

Patient_No	drugNo	no_of_units	patient_name	drug_Name
P_1	d_1	10	Ajith	Crocin
P_1	d_2	15	Ajith	dolo
P_2	d_2	20	Rama	dolo

(The redundancies are circled)

FDS:

{patient No, drug No} → no of units (FFD)

Ck

patient No → patient Name (PFD)

drug No → Drug Name (PFD)

Now we decompose the relation to eliminate redundancy as shown below

Patient_No (FK)	drug No (FK)	no of units
P_1	d_1	10
P_1	d_2	15
P_2	d_2	20

{Patient_No, drugNo} → no_of_units

Patient No	Patient_Name
P_1	Ajith
P_2	Rama

Patien_No → patient_Name

Drug No	Drug Name
d_1	Crocin
d_2	dolo

Drug No → Drug Name

→ Generally in R(ABCD)

If AB is ck and F:{AB→C, A→D}.

for every PFD we create a seperate table.

for all FFD we put them in the same table

Decomposition is

$R_1(ABC)$ $R_2(AD)$

Every decomposition of this kind is lossless and dependency preserving

Eg 2: Consider R(ABCDE)

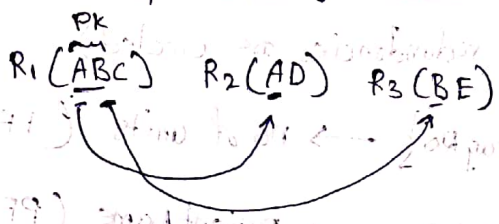
F: {AB→C, A→D, B→E}

ck: AB

A→D, B→E are PFDs

R is in INF

now we decompose it is

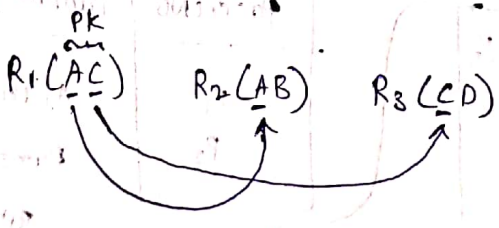


Eg 2: R(ABCD)

F: {A→B, C→D}

ck: AC

decomposition is.



* Here if we decompose into two tables only $R_1(AB)$ $R_2(CD)$ then the decomposition will be lossy.

Eg: R(ABCDE)

F: {AB → C, A → D, D → E}

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

A → D is a PFD

∴ Decomposition is

R_1(A D E) R_2(A B C)

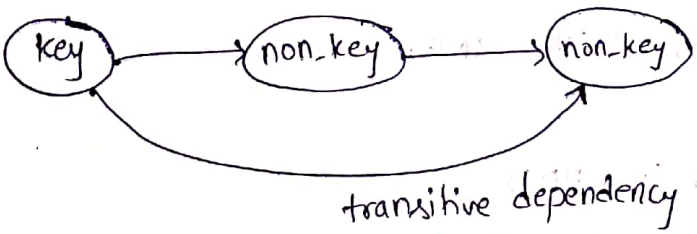
Note:

A relation with key of one attribute is always in 2NF.

3NF:

→ 3NF should first be in 2NF

→ 3NF not allows transitive dependencies.



Transitive dependency exists whenever there is non-key → non-key dependency. (or) {Part of key + non-key} → Non-key

Eg: Employee(Eno, Ename, seniority, salary)

Eno → Ename, seniority,

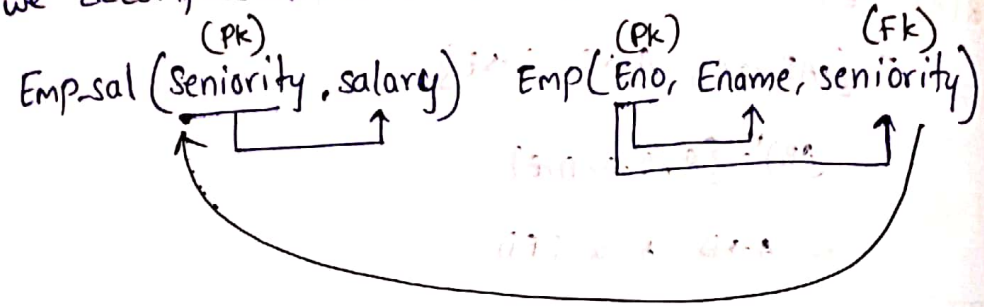
seniority → salary

Ck: Eno

It has no PFDs.

∴ It is in 2NF but not in 3NF.

So we decompose the relation as



This decomposition is also lossless always.

Eg: R(ABC)

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

R is in 2NF but not in 3NF

∴ It is decomposed as

$R_1(AB)$ $R_2(BC)$

Eg: R(ABCDE)

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

Decompose R into 3NF.

Sol:

$B \to D$ is : PFD

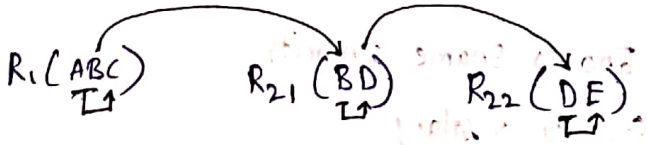
∴ R is not in 2NF

Decompose into 2NF

$R_1(ABC)$ $R_2(BDE)$

R_1 is in 3NF R_2 is in 2NF but not in 3NF

∴ Decomposition is



* → Most database use 3NF and it is considered adequated.

Algorithm to decompose into 3NF: (from 1NF to 3NF)

This decomposition is lossless and dependency preserving.

Step1: Find minimal set of F called G.

Step2: Take each FD into a seperate relation

This step ensures that the decomposition is dependency preserving

Step3: If no decomposed relation contains key then take key attributes into seperate table (To ensure lossless decomposition)

Step4: If there are any redundant relations, then eliminate them.

Eg: $R_1(AB)$ $R_2(ABC)$

Here R_1 is redundant because attribute A, B are already present in R_2.

- Eg: R(ABCD)

F: {AB → C, B → BD}

Step 1: F is minimal

Ck: AB

Step 2: Create individual relations for each FD.

∴ $R_1(ABC)$ $R_2(BD)$

Step3: R_1 has key

Step4: No redundant relations

∴ $R_1(ABC)$ $R_2(BD)$ is in 3NF

Eg: R(ABCD)

F: {A → B, C → D}

Ck: AC

Step 1: F is minimal

Step 2: $R_1(AB)$ $R_2(CD)$

Step 3: There is no relation with key attributes.

So we create one

i.e., $R_3(AC)$

Step 4: $R_1(AB)$ $R_2(CD)$ $R_3(AC)$

no redundant relations

∴ 3NF

Eg: R(ABCDE)

F: {AB → C, B → D, D → E}

Ck: AB

Step 1:

F is minimal

Step 2: $R_1(ABC)$ $R_2(BD)$ $R_3(DE)$

Step 3: R_1 has key attribute

Step 4: no redundant relations

∴ $R_1(ABC)$ $R_2(BD)$ $R_3(DE)$

is in 3NF.

P/3

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

AB is key

Decomposing into 3NF:

$R_1(ABC)$ $R_2(ADE)$ $R_3(BF)$ $R_4(FGH)$ $R_5(DIJ)$

R_1 has key attributes

Also there are no redundant relations

∴ It is in 3NF

2NF decomposition:

$R_1(ABC)$ $R_2(ADEIJ)$ $R_3(BFGH)$

(P/4)

Step(i):

F is minimal

Step(ii)

$R_1(ACF)$ $R_2(AB)$ $R_3(BE)$ $R_4(CD)$

Ck: $(AC)^+ = \{A, B, C, D, E, F\}$

Step(iii):

key attributes present in R_1

Step(iv):

no redundant attributes

∴ $R_1(ACF)$ $R_2(AB)$ $R_3(BE)$ $R_4(CD)$

(P/5)

$A^+ = [A, F, C, D\}$

$B^+ = [B, E\}$

∴ Ck: AB

$R_1(AFC)$ $R_2(CD)$ $R_3(BE)$

no relation has all key attributes

∴ $R_1(AFC)$ $R_2(CD)$ $R_3(BE)$ $R_4(AB)$

Eg: R(ABC)

F: {AB → C, C → A}

→ Prime transitivity

$(AB)^+ = [A, B, C]$.

$C^+ = [A, C]$

Ck: AB, CB

~~There is PFI~~

There is no PFD

There is no transitive dependency

∴ This is 3NF

Note:

→ If a relation schema R is in 3NF then for every dependency X → A ~~one of~~ atleast one of below two conditions holds

(i) X is super key

(ii) A is prime attribute.

∴ If all attributes are prime attribute then the relation is in 3NF

Eg: R(ABCD)

F: {AB → C, B → D, D → B, C → A}

Find normal form of R

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

Ck: AB, AD, CB, CD

all are prime attributes

∴ 3NF

BCNF (Boyce- codd Normal form):

→ The problem with 3NF is prime transitivity.

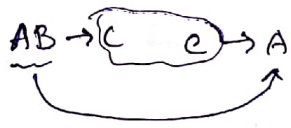
Eg: Consider R(ABC)

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

CK: AB, CB

R is in 3NF

Considering the information from FDs alone, BCNF has 0 redundancy



Here prime attribute A is depending on CK AB through another prime attribute C.

This is called prime - transitivity

This prime transitivity is not allowed in BCNF.

AB	C
c_1	a_1
c_1	a_1
c_1	a_1

$C \to A$

Redundancy due to prime transitivity

→ A relation schema 'R' is in BCNF if whenever a nontrivial FD of form $X \to A$ holds,

X must be super key

If R is a relation and $X \to A$ is dependency violating BCNF property then we decompose into $R_1(R-A)$ $R_2(XA)$ and continue this process recursively

Eg: R(ABC) $F = \{AB \to C,\ C \to AB\}$

CK: AB, C

In $AB \to C$, AB is super key.

$C \to AB$, C is super key

∴ R is in BCNF

Eg: R(ABC) F: {AB → C, C → A}

Ck: AB, CB

In C → A, C is not superkey

R is not in BCNF but in 3NF

So we need to decompose

since C → A is violation

create R(AC)

to make sure it is lossless we

create anotte relation R(AB) (or) R(BC)

(∵ B is remaining attribute)

$R_1(\underline{AC})$ $R_2(AB)$ (or) $R_1(\underline{AC})$ $R_2(BC)$

Both these are not dependency preserving

Note:

→ All key relation is a relation formed by attributes of one candidate key

→ Every all key relation is always in BCNF

∴ $R_2(BC)$ is in BCNF

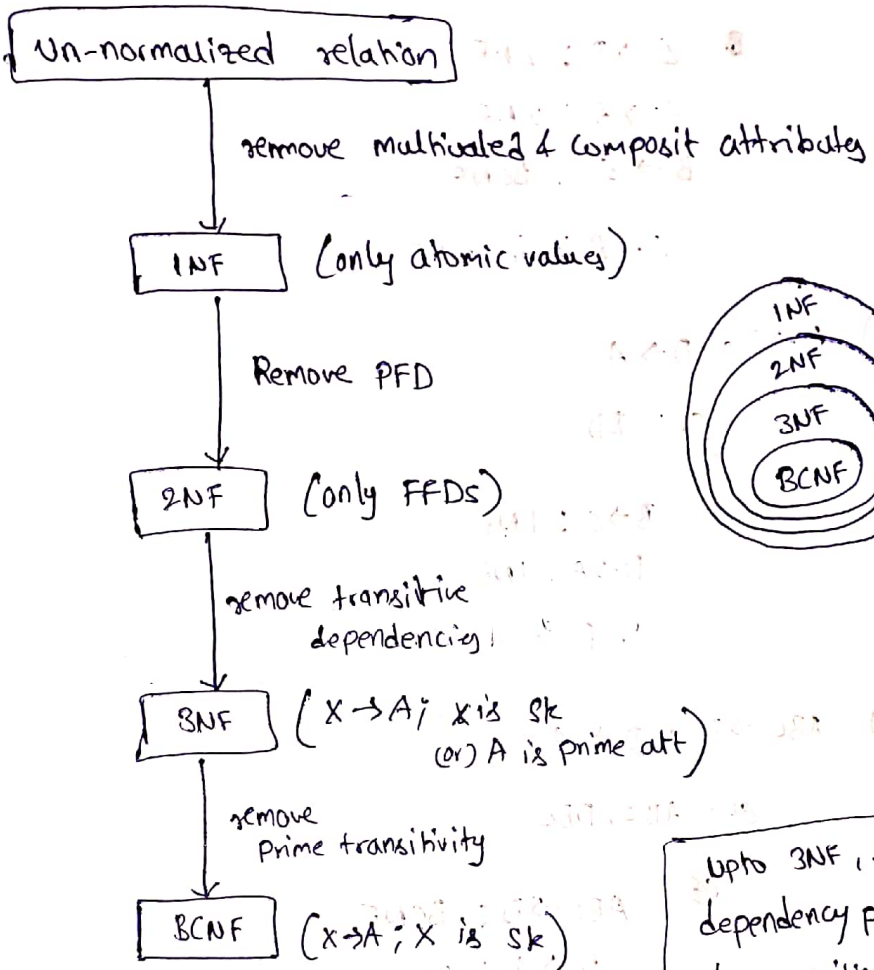
→ Sometimes lossless and dependency preserving decomposition may not be possible for every relation.

Thus 3NF is mostly used.

Also for higher normal forms more no of relations are formed and thus accessing data may be slow.

→ Any relation with two attributes is in BCNF.

Note:



Upto 3NF, lossless, dependency preserving decompositions are possible. But for BCNF it may or may not be possible

~~Note:~~

~~→ If there is (part of key → non-key) then 1 NF~~

~~• else if there is~~

Note:

* When asked to find normal form of a relation we start ~~from~~ backwards

(i) If every dependency is of form

Sk → any	: BCNF
Sk → PA	: 3NF
Nk → Nk	: 2NF
Part of key → Nk	: 1NF

(P16) R(ABCD)

i) $C \to D$; $C \to A$; $B \to C$

Ck : B

$C \to D$: 2NF

$C \to A$: 2NF

$B \to C$: BCNF

∴ R is in 2NF

2) $B \to C$; $D \to A$

Ck : BD

$B \to C$: 1NF

$D \to A$: 1NF

∴ R is in 1NF

3) $ABC \to D$; $D \to A$

Ck : ABC, DBC

$ABC \to D$: BCNF

$D \to A$: 3NF

∴ R is in 3NF

4) $A \to B$, $BC \to D$, $A \to C$

Ck : A

$A \to B$: BCNF

$BC \to D$: 2NF

$A \to C$: BCNF

∴ R is in 2NF

5) $AB \to C$, $AB \to D$, $C \to A$, $D \to B$

Ck : AB, CB, AD, CD

all are prime attributes

∴ 3NF

$C \to A$: 3NF

∴ R is in 3NF

(P7) R(ABC)

F_1: {A→B, B→C, C→AB}

CK: A, C, B

A→B: BCNF

B→C: ~~2NF~~ BCNF

∴ R_1 is in ~~2NF~~ BCNF

(P8) R(PQRS)

F: {QR→S, R→P, S→Q}

Y(PR) Z(QRS)

F_1: R→P

P^+ = {P}

R^+ = {R, P}

F_2: QR→S

S→Q

Q^+ = {Q}

S^+ = {S, Q}

R^+ = {R, P}

Y is in BCNF

F_2: {QR→S, S→Q, SR→Q}

CK: QR, SR

S→Q: 3NF

∴ Z is not in BCNF

R is common attribute

R^+ in F = {R, P}

Y(PR)

∴ lossless decomposition

$F_1 \cup F_2$: {R→P, QR→S, S→Q, SR→Q}

∴ dependency preserving

∴ opt (a)

P/9 client_id → f_name, L_name, ~~city, zip_code~~

order_id → order_data, city, zip_code

∴ ~~PFD is present~~ PFDs present

∴ relation in INF

∴ violating 2NF

→ when asked to find a normal form of relation we find highest normal form.

when asked to find violating normal form; we find lowest normal form.

P/10

cid, sid → bill

sid → store_loc

↳ PFD

∴ take it to seperate table and sid pk and sid Fk in original table

∴ opt ©

P/11

~~$(F_1)^+ = F_1, F_3$~~

F_1, F_2 → independent

$\therefore (F_1F_2)^+ = \{F_1, F_2, F_5, F_3, F_4\}$

CK: F_1F_2

$F_1 \to F_3$: ~~2NF~~ 1NF

$F_2 \to F_4$: ~~2NF~~ 1NF

$F_1F_2 \to F_5$: BCNF

∴ opt (a)

17/11/20

P/13

a) $B \to C$

$AB \to C$

Ck: AB

~~∴ BCNF~~ ∴ 1NF

b) $AC \to B$

$B \to C$

Ck: AC, AB

all are prime

∴ 3NF

c) $A \to B$

$B \to C$

Ck: A

$A \to B$: 3NF

$B \to C$: 2NF

∴ 2NF

d) $C \to AB$

$B \to C$

Ck: C, B

∴ BCNF

P/15

$AB \to D$

$BC \to D$

$A \to C$

$C \to A$

$(AB)^+ = \{ABCD\}$

Ck: AB, CB

∴ 3NF

P/16

I. BCNF

II. Ck: {rno, cid}, {email, cid}

∴ 3NF but not BCNF

Relational Algebra:

→ procedural language that give query evaluation plan.

1) Selection (σ):

This operator is used for ~~section~~ selection of rows.

2) projection (π):

This operator is used for selection of columns

Eg: Consider below student relation

RNo	Name	marks
1	Ajay	15
2	shiva	16
3	Ajay	13
4	Ramu	9

Query: find name of the students whose marks are greater than 10.

$$\pi_{Name}\left(\sigma_{mark>10}(Student)\right)$$

```
select name
from student
where marks >10;
```

O/P:

Name
Ajay
shiva

(Here ∵ Ajay is printed only once)

Note:

→ RA & RC has duplicate elimination implicitly

→ But SQL gives duplicates also

→ So we need to use distinct keyword in SQL.

Set manipulation operators:

∪ - Union

∩ - Intersection

(−) - Difference

To perform any of ~~these~~ these operations blw two sets, the sets must be compatible

Two relational instances are said to be compatible iff

(i) same no of colums and

(ii) corresponding attributes must have same domain

Eg: Consider

Depositor

cname	AcnO
Ajay	1
Balu	2
Charan	3

Borrower

Cname	Loan no
Devi	9
Ajay	10
Sita	11

Query: find name of the customers who have an account (or) loan (or) both

$$\left\{ \left(\Pi_{\text{cname}} \text{ Depositor} \right) \cup \left(\Pi_{\text{cname}} \text{ Borrower} \right) \right\}$$

Here we don't need to select any rows. So we don't need to use σ.

O/P:

Cname
Ajay
Balu
Charan
Devi
Sita

Query: Find customer names who are depositors but not borrowers.

$$\{(\Pi_{Cname}\ Depositor) - (\Pi_{Cname}\ Borrower)\}$$

Cross Product: (x)

It is useful when we need to process data from more than one tuple.

Consider

R

A	B
1	2
3	4

S

C	D
5	6
7	8

$\Rightarrow$

R x S

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

(order of attribute might be ABCD silly in SXR it is CDAB

Eg:

Borrower

Cname	Lno
Ajay	9
Sita	10

Loan

Lno	Branch	Amount
9	ABIDS	10k
10	Sec	9k

Query: find name of the customer who have a loan in the branch abids.

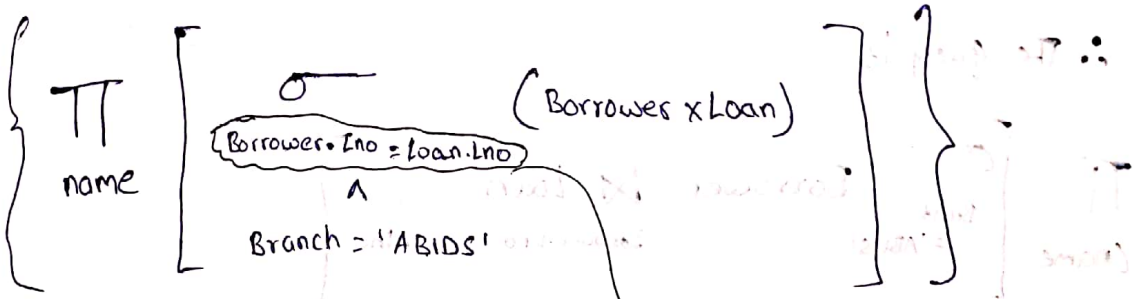
→ For queries involving 2 relations we perform cartesian product.

Borrower x Loan

Cname	Lno	Lno	Branch	Amount
Ajay	9	9	ABIDS	10k
Ajay	9	10	Sec	9k
Sita	10	9	ABIDS	10k
Sita	10	10	Sec	9k

Actually when a name conflict occur while performing cross product, no names will be given and we need to refer them by position

Here combinations are where Lno are equal



∴ O/P :

Name
Ajay

Join condition : Condition used when combining two relations.

18/11/20

JOIN: (⋈)

It is a catesian product followed by a selection and a projection

Consider R(A B) S(B C)

Here $R \underset{R.B=S.B}{\bowtie} S = \underset{A,B,C}{\Pi} \left(\underset{R.B=S.B}{\sigma} R \times S \right)$

Consider the previous query of finding customer who have loan in abids

$$\text{Borrower} \bowtie \text{Loan}$$

Borrower.Lno = Loan.Lno

Cname	Lno	Branch	Amount
Ajay	9	ABIDS	10k
Seta	10	Sec	9k

∴ The query is

$$\Pi_{\text{Cname}} \left[\sigma_{\text{Brand} = \text{'ABIDS'}} (\text{Borrower} \bowtie_{\text{Borrower.Lno} = \text{loan.Lno}} \text{Loan}) \right]$$

o/p:

Cname
Ajay

Natural join:

In natural join, the join condition is by default equality on all common attributes (i.e., attributes with same name)

R	
A	B

S	
B	C

$$R \bowtie S = R \bowtie_{R.B=S.B} S = \Pi_{A,B,C} \left(\sigma_{R.B=S.B} (R \times S) \right)$$

↓ Natural join

Now the previous query can be written as

$$\Pi_{\text{Cname}} \left[\sigma_{\text{Branch} = \text{'ABIDS'}} (\text{Borrower} \bowtie \text{Loan}) \right]$$

However, we can write this query in efficient way.

$$\Pi_{Cname}\left[\text{Borrower} \bowtie \left(\sigma_{Branch='ABIDS'}\ Loan\right)\right]$$

Here no of tuples joined will be reduced by a large factor.

This query requires less buffer size and less no of comparisions.

Normal query : Join & select

Optimized query : select & join

→ Also while joining, performing join on smaller tables first is (more than 2 tables) more efficient

(P/8) Here we have

branch ⋈ (account ⋈ depositor)

to optimize we perform select first and then join

$$\left(\sigma_{\substack{b\text{-}city \\ ='Agra'}}\ branch\right) \bowtie \left[\left(\sigma_{balco}\ account\right) \bowtie depositor\right]$$

to optimize further we can first joint branch & account

∵ account & depositor are bigger relations

$$\therefore \left(\sigma_{\substack{b\text{-}city \\ ='Agra}}\ branch \bowtie \sigma_{balco}\ account\right) \bowtie depositor$$

But from the given options, opt @ is the best.

P/4

For every tuple in S, we have a tuple in R such that S.B = R.B

∴ In R×S we get 100 correct combinations

∴ 100 tuples

This maximum possible

i.e., if there is R.B for every S.B

P/5

Here we have exactly 2000 since R in π_1 references R in π_2

Note:

whenever join is performed blw parent & child, the no of tuples = no of tuples in child.

P/11

$$\Pi_X\left(\sigma_{\substack{P.Y=R.Y \\ \wedge \\ R.V=V2}}(P \times R)\right) = \{(x_2)\}$$

$$\Pi_X\left(\sigma_{\substack{Q.Y=R.Y \\ \wedge \\ Q.T>2}}(Q \times R)\right) = \{(x_1)\}$$

$$\therefore \{(x_2)\} - \{(x_1)\} = \{(x_2)\}$$

∴ 1

(P/9) $\Pi_{cid}\left[\text{enroll} - \left\{\left(\Pi_{sid}\left(\sigma_{sex='female'}(\text{studinfo})\right)\right) \times \Pi_{cid}(\text{enroll})\right\}\right]$

$\Pi_{sid}\left(\sigma_{sex='female'}(\text{studinfo})\right)$: gives all ids of female students

$\Pi_{cid}(\text{enroll})$: gives all courses enrolled by atleast one student

$\left(\Pi_{sid}\left(\sigma_{sex='female'}(\text{studinfo})\right)\right) \times \Pi_{cid}(\text{enroll})$: gives all possible combinations of female sids and cids

Now removing this from enroll have no affect on tuples involving males.

∴ It gives all courses in which male students enrolled.

(P/10) $\Pi_{name}\left(\sigma_{sex=female}(\text{Student})\right)$

: gives names of all female students.

$\Pi_{name}\left[(\text{student}) \bullet\!\!\bowtie_{\substack{sex=female \\ \wedge x=male \\ \wedge marks \le m}} \rho_{n,x,m}(\text{student})\right]$

: give names of all female students whose marks are less than or equal to atleast one boy.

∴ {all female} − {female whose marks ≤ atleast one boy's marks}

∴ All females whose marks greater than all the boys.

Division (÷)

This used to compare a tuple of a relation with all the tuples of other relation.

Eg: Consider two relation A(X,Y) B(Y)

A ÷ B contains all X value such that X is related to ~~exa~~ every value of Y in B.

Eg:

A	
X	Y
x_1	y_1
x_1	y_2
x_1	y_3
x_2	y_1
x_2	y_2
x_3	y_1

B
Y
y_1
y_2
y_3

⇒

A ÷ B
X
x_1

Eg: Consider below relations

Student

Sid	name
1	Ajay
2	Rama

Enroll

sid	cid
1	9
1	10
2	10

Course

cid	cname
9	DB
10	DS

Query: find name of student who has enrolled for all the courses

Sol:

Comparing with the previous example here we have

X: Name Y: Courseid

So we need to create two relation A(name, cid) B(cid) and perform division

$$\left[\therefore \Pi_{name,cid}(Student \bowtie Enroll)\right] \div \left[\Pi_{cid}(Course)\right]$$

name	cid
Ajay	9
Ajay	10
Rama	10

÷

cid
9
10

↓

~~cid~~	name
	Ajay

One more of writing query is the same

$$\Pi_{name}\left\{Student \bowtie \left[Enroll \div \left(\Pi_{cid}(Course)\right)\right]\right\}$$

Relational Calculus:

→ It is a non-procedural language (i.e., it doesn't give the process)

→ It is called declarative language.

→ It describes result of query but not the step by step procedure.

→ two types:

(i) Tuple Relational calculus (TRC)

(ii) Domain Relational calculus (DRC)

→ In TRC, result is described as set of tuples.

→ In DRC, result is described as set of values of an attribute.

TRC	DRC
→ result is described as set of tuples	→ result described as set of values of an attributes
→ set of tuples are represented as tuple variable	→ set of values are represented as domain variable
→ form of TRC query $\{T \mid P(T)\}$ ↓ tuple variable ↓ Formula that describes T	→ form of DRC query $\{\langle d_1, d_2 \cdots d_n \rangle \mid P(d_1, d_2, \cdots d_n)\}$ ↓ domain variables ↓ Formula that describes domain variables

Eg: Consider below relations

Student (Rno, name, marks)

Enroll (Rno, cno)

Course (cno, cname, duration)

> In TRC, for a variable R, if we don't write ∃R ∈ relation, then the convention is that R has all the fields of relations that appear in the formula

Q1: Find name of the student whose marks > 10;

TRC: $\{T / \exists S \in student\ (S.marks > 10 \wedge T.name = S.name)\}$

- T: tuple variable
- S: tuple variable
- T.name: This attribute can be given any name (for display)

DRC: $\{\langle N \rangle / \exists R, M\ [\langle R, N, M \rangle \in student\ (M > 10)]\}$

- N: Domain variable representing name
- R: rno
- M: marks

Q2: Find name of the student who enrolled a course, with ~~Course~~ cno = 9.

TRC: $\{T / \exists S \in student\ (T.name = S.name) \wedge \exists E \in Enroll\ (E.cno = 9 \wedge S.Rno = E.Rno)\}$

DRC: $\{\langle N \rangle / \exists R, M [\langle R, N, M \rangle \in student] \wedge \exists C\ [\langle R, C \rangle \in Enroll (C = 9)]\}$

Here R need not to be mentioned again

Here we need value of 'c' to be 9. So this query can also be written as

$$\{\langle N\rangle \mid \exists R, M[\langle R, N, M\rangle \in \text{student}] \wedge [\langle R, 9\rangle \in \text{Enroll}]\}$$

~~when do~~

Q3: find name of the student and name of the course who enrolled a course of duration 3 months.

TRC: $\{T \mid \exists S \in \text{Student}(T.\text{name} = S.\text{name}) \wedge \exists C \in \text{Course}(C.\text{duration} = 3 \wedge T.\text{cname} = C.\text{cname}) \wedge \exists E \in \text{Enroll}(E.\text{rno} = S.\text{rno} \wedge E.\text{cno} = C.\text{cno})\}$

DRC: $\{\langle N, C\rangle \mid \exists R, M[\langle R, N, M\rangle \in \text{student}] \wedge \exists cno, duration[\langle cno, C, dur\rangle \in \text{Course}($

$\{\langle N, C\rangle \mid \exists R, M[\langle R, N, M\rangle \in \text{student}] \wedge \exists cno, C, 3[\langle cno, C, 3\rangle \in \text{Course}] \wedge [\langle R, cno\rangle \in \text{Enroll}]$

Eg: What does below query return

$$\{T \mid \exists P \in professor\ (T.name = P.name) \wedge \exists C \in Course\ (C.cname = 'DBMS' \wedge P.SSN = C.SSN)\}$$

Sol:

It returns names of professors who teach DBMS

A Query language than can express all the queries possible with relational algebra, is said to be relationally Complete

Unsafe Query:

$\{T \mid T \in R\}$ returns finite no of tuples

$\{T \mid \sim(T \in R)\}$ return infinite no of tuple

→ A query that returns infinite no of tuples is called unsafe query.

→ Relational algebra has only safe queries.

Relational algebra $\cong$ Safe relational calculus

Q 4-07 Consider the relation employee(name, sex, supervisorName).

What does below query produce.

$$\{e.name \mid employee(e) \wedge \forall x [\sim employee(x) \vee x.supervisorName \neq e.name \vee x.sex = \text{"male"}]\}$$

Now we write it as

$$\{ e.name \mid employee(e) \wedge \sim \exists x [employee(x) \wedge x.supervisorName = e.name \wedge x.sex \neq \text{"male"}] \}$$

$$= \text{"female"}$$

Structured Query Language

Order of evaluation:

(i) from class

If we write more than one table in from class, cartesian product is performed.

(ii) where

selection of rows is done

(iii) group by

divide rows into groups

(iv) HAVING

select the formed groups

(v) Expressions in select class are evaluated

(vi) distinct in select

(vii) order by

sort the rows

After this the o/p is produced

Simple Select:

> select name from student;

> select name, branch from student;

> select * from student;

> select rno, marks+5 from student; // Mathematical expressions are also possible.

> select distinct name from student;

select with where:

where is used for selecting rows based on some conditions.

→ The conditions may be connected by below operators if needed

- → and, or, not
- → in, not in
- → between .. and
- → not between .. and
- → is null, is not null
- → <, <=, >, >=, <>, !=, =
  - ↳ not equal
- → Like '%', Like '_'

'!=' is not there in SQL

Eg: Consider

Student(Rno, name, branch, year, marks, passport)

Strings in SQL are enclosed within single quotes

```
> select *
  from student
  where branch = 'CSE' and year = '1';
```

O/P:
It display all students of CSE studying 1st year.

The evaluation is done row by row starting from the 1st row

> ==

where not branch = 'CSE';

(or)

<> ==

where branch <> 'CSE';

} gives all student records who are not CSE.

in operator: (set comparision operator)

where branch = 'CSE' or branch = 'IT' or branch = 'ME'

the above can be written as

where branch in ('CSE', 'IT', 'ME');

not in:

Opposite of in operator

where branch not in ('CSE', 'IT', 'ME');

* gives students of branches other than CSE and IT and ME

between ... and:

where marks between 10 and 20; (inclusive)

≅ where marks ≥ 10 and marks ≤ 20

Similarly

where marks not between 10 and 20; (exclusive)

≅ where marks < 10 or marks > 20

is null & is not null

To check if the value is present or not.

where passport is null; → students who don't have passport

where passport is not null; → students who have passport.

Like:

→ used for pattern ~~matth~~ matching

'%' → matches to '0' or more characters

'_' → matches to one character.

Eg: where name like 'R%'; // student names starting with R

where name like '%H'; // student names ending with H

where name like '%S%'; // containing S

where name like '___'; // names of length 3

~~when~~

* 'R__%' // names of length atleast 3 and starting with R.

Not:

> select *
from student
where 1=2;

} This kind of queries are valid and it returns 0 rows.

> where 1<2 ----- all rows are printed in o/p

> select *
from student
where NULL=0;

} NULL is a keyword and it is not zero
∴ 0-rows will be returned

> ~~select~~
where NULL<>0; --- return 0 rows
(comparing null with any value is always false) (∵ null is unknown)

> where NULL=NULL; --- returns 0 rows.

> where NULL is NULL ; .... returns all the rows.

> where NULL <> NULL ; ... returns 0 rows

→ Comparision of NULL with any value using relational operator is false.

→ Comparision of NULL with 'is NULL' is only true.

Aggregate Functions:

~~SQL has 5~~

→ Aggregate functions take a set of values and returns one value.

→ SQL has 5 aggregate functions

(i) Min
(ii) Max
(iii) Sum
(iv) Avg
(v) Count

Eg: Student

Name	M1	M2
A	10	5
B	null	10
C	20	15
D	null	20
E	30	null
F	null	25

Note:
Aggregate functions ignore null value

(i) select min(M1) from student;

o/p: 10

(ii) select max(Name) from student;

o/p: F

MIN, MAX (COUNT) can take input either text or number

(iii) Select sum(m1) from student;

o/p : 60

(iv) select sum(m1,m2) from student;

This gives error

→ aggregate func takes only one set as i/p

(v) select sum(m1+m2) from student;

m1+m2 is single set . So no error.

Any mathematical operation with null gives null

null + any value is null

(∵ unknown + any value is unknown)

(15 + null + 35 + null + null)

∴ o/p : 50

(vi) select sum(m1) + sum(m2) from student

sum(m1) = 60

sum(m2) = 75

∴ o/p : 135

(vii) select count(m1) from student

o/p : 3

(viii) select count(*) from student;

o/p : 6 (i.e. no

Aggregate functions with DISTINCT

(i) COUNT(DISTINCT A) :
no of distinct values in column A.

(ii) SUM(DISTINCT A) :
Sum of all unique values

(iii) AVG(DISTINCT A)
Avg of all DISTINCT values

* Using DISTINCT with MIN & MAX doesn't make any sense

(ix) select avg(m1) from student;

o/p : 20 (i.e., $\frac{60}{3}$)

→ $avg(m_1) = \frac{sum(m_1)}{count(m_1)}$.

avg & sum. works only only on numerical data

(x) select sum(name) from student;

→ Error

(xi) select name from student
where m1 = max(m1);

o/p:

This gives error.

(∵ where is a row function (i.e., it selects row) and max is a group function (i.e., max considers are all column) so performing this is difficult for query processor)

Note:

→ Using aggregate functions in where clause give error.

→ However we can use aggregate function in 'Having' class.

Set manipulation Operators

- we have 3 set manipulation operators

R { Union, interset, except } S

↳ difference

Set manipulation operators eliminates duplicates.

R & S must be compatible.

Order by:

used to sort the rows

→ ASC
→ DESC

Eg: Company

Cno	Name	turnover
1	B	5Cr
2	A	3Cr
3	B	4Cr
4	C	6Cr
5	B	5Cr

Query: Display all details of company in order of their name

```
> select *
  from company
  order by name ASC;
```

Silly for descending order we use DESC

Query: Display company details in descending order of their name and increasing order of their turn over

```
> select *
  from company
  order by name desc, turnover asc;
```

o/p :

Cno	cname	turnover
4	C	6Cr
3	B	4Cr
1	B	5Cr
5	B	5Cr
2	A	3Cr

While sorting if the attribute values are same, then ordering will be based on insertion i.e., first inserted records will come first

Group by and Having clause:

→ Group by used for grouping rows and 'having' specifies grouping conditions.

Eg:

Student

Rno	Name	Branch	year
1	A	CS	I
2	B	CS	I
3	C	CS	II
4	D	IT	I
5	E	IT	I
6	F	ME	II

Q: Find no of students in each branch.

```
>select branch, count(*)  -- (3)
 from student             -- (1)
 group by branch;         -- (2)
```

} order of execution

O/P:

Branch	count(*)
CS	3
IT	2
ME	1

Q: Find no of student in each branch and year.

```
> select branch, year, count(*)
  from student
  group by branch, year;
```

o/p:

branch	year	count(*)
CS	I	2
CS	II	1
IT	I	2
ME	II	1

Note:

Consider below query

```
> select branch, year, count(*)
  from student
  group by branch;
```

This will result an error.

Note:

Every attribute of select clause must be either present in group by clause and other aggregate functions can also appear in select clause.

Q: Find no of students in each branch if the branch contains atleast two students.

```
> select branch, count(*)  ....(4)
  from student             ....(1)
  group by branch          ....(2)
  having count(*) >= 2;    ....(3)
```

Conditions of group by are specified using having clause

o/p:

branch	count(*)
CS	3
IT	2

Q: Find the no of students in each group other than cs.

```
> select branch, count(*)
  from student
  group by branch
  having branch <> 'cs';
```

Note:

Having clause contains either aggregate functions or attributes that appeared in group by.

Where vs Having

→ where selects rows

having selects groups

→ aggregate functions cannot be used in where

aggregate functions can be used in having.

20/11/20

* Q: What is the o/p of the following query based on the above-student table.

```
> select count(*)
  from student
  having count(*) >= 3;
```

a) error b) 6 c) 0 d) unpredictable o/p.

Sol:

Since group by is not specified, entire table is considered one group.

(count*) = 6 >= 3

∴ O/P : count(*)
6

∴ opt (b)

Note :

→ A query containing having clause without group by, treats whole table as a single group.

P/7 A, B and aggregate function can appear in select clause

∴ opt (b)

P/8 If a is key

count(a) > 1 will always be false and no o/p will be given.

∴ a shouldn't be key

∴ opt (c)

JOIN :

Consider

professor(SSN, name, dept)

course(cno, cname, ssn)

Q : find name of the professor teaching database.

```
> select name
  from professor, course        → Cartesian Product
  where professor.SSN = course.SSN   and   cname = 'Database';
        ↓ join condition                   ↓ selection condition
```

→ Here we are first performing join and then selecting rows. This is inefficient (for large tables)

i.e., we are specifying join condition and selection condition in the same where clause.

join..on :

→ ~~To remove the previous discussed inefficienty we use join ... on~~

→ Now the same query is written as

```
> select name
  from professor join course on professor.ssn = course.ssn
  where cname = 'Database';
```

This query, interms of efficiency and evaluation is same as that of the previous query but it is more readable.

Natural join :

→ Equality is implicity specified on ~~equal~~ same named fields

Now the same query can be written as

```
> select name
  from professor natural join course
  where cname = 'Database';
```

Q 6-11

Database table by name loan_records is is given below

Borrowers	Bank_Manager	Loan_amount
Ramesh	Sunderajan	10000
Suresh	Ramgopal	5000
Mahesh	Sunderajan	7000

what is the o/p of the following SQL query?

```
SELECT count(*)
from (
Select borrower, bank_manager from loan_records) as s
Natural join
(select bank_manager, loan_amount from loan_records) as T.
```

Sol:

o/p of 1st sub query

Ramesh	Sunderajan
Suresh	Ramgopal
Mahesh	Sunderajan

o/p of 2nd sub query

Sunderajan	10000
Ramgopal	5000
Sunderajan	7000

∴ The natural join is

Ramesh	Sundejan	10000
"	"	7000
Suresh	ramgopal	5000
Mahesh	Surderjan	10000
"	"	7000

∴ 5

Outerjoin:

It is used to join one relation (all rows) even if there is no matching row.

Eg: Consider

Professor

SSN	Name
1	A
2	B
3	C
4	D

Course

cid	Cname	SSN
9	DB	1
10	DS	3
11	CD	1
12	CO	NULL

Q: find name of all the professors and name of the courses that they teach if any.

↳ i.e., even if a professor doesn't teach any course, we need to display his name.

In situations like this we use outer join.

→ Based tuple of which relation we need completely, outer join is of 3 types:

(i) Left outer join (keyword: left join)

(ii) right outer join (keyword: right join)

(iii) Full outer join (keyword: full join)

Consider

> seled name, cname
from professor natural join course;

o/p:

name	cname
A	DB
A	CD
C	DS

Here we don't get professor names who doesn't teach any course.

So we use left-outer join as shown below

> select name, cname
from professor natural left join course;

o/p:

name	cname
A	DB
A	CD
C	DS
B	null
D	null

Note:

The same above query can be written as shown below

> select name, cname
from professor, course
where professor.SSN = course.SSN(+);

(+) says course.SSN is optional.

Q: Consider

R

A	B
a_1	b_1
a_2	b_1
a_1	b_3
a_2	b_4
a_3	b_5

S

B	C
b_1	c_1
b_2	c_2
b_3	c_2
b_4	c_1

Find o/p for each of the below queries

(i) > select *
from R natural join S;

A	B	C
a_1	b_1	c_1
a_2	b_1	c_1
a_1	b_3	c_2
a_2	b_4	c_1

no of rows = 4

(ii) > select *
from R natural left join S;

A	B	C
a_1	b_1	c_1
a_2	b_1	c_1
a_1	b_3	c_2
a_2	b_4	c_1
a_3	b_5	

no of rows
4+1=5

(iii) > select *
from R natural right join S;

A	B	C
a_{11}	b_1	c_1
a_2	b_1	c_1
	b_2	c_2
a_1	b_3	c_2
a_2	b_4	c_1

no of rows
$= 4 + 1 = 5$

(iv) select *
from R natural full join S;

A	B	C
a_1	b_1	c_1
a_2	b_1	c_1
a_1	b_3	c_2
a_2	b_4	c_1
	b_2	c_2
a_3	b_5	

no of rows
$= 4 + 1 + 1 = 6$

Note:

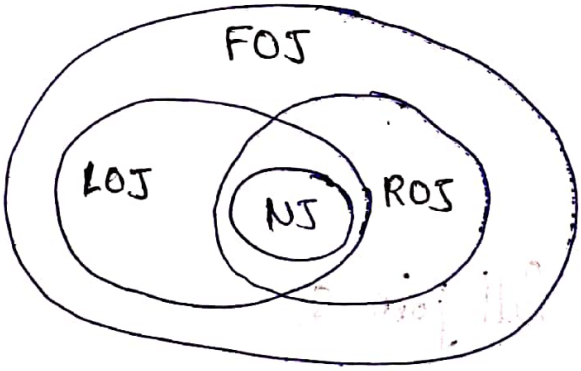
→ no of rows in natural join
= no of matching rows

→ no of rows in a left outer join
= no of matching rows + no of non-matching row for left table

→ no of rows in right outer join
= no of matching rows + no of non-matching rows for right table

→ no of rows in full outer join

= no of matching rows +
no of non-matching rows of left table +
no of non-matching rows of right table.

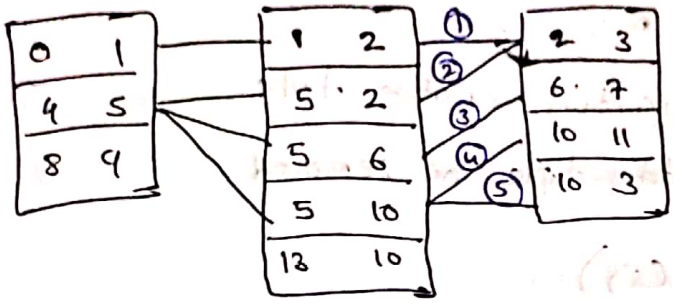


Set of tuples returned by NJ $\subseteq$ set of tuple returned by LOJ

Set of tuples retured by NJ $\subseteq$ set of tuples returned by ROJ

$$\left\{\begin{array}{c}\text{Set of tuples by LOJ}\\ \cup \\ \text{set of tuple by ROJ}\end{array}\right\} = \{\text{set. of tuples of FOJ}\}$$

P/9



∴ 5 rows

P/13

R

A	B	C
1	5	7
3	7	
4		9

∴ opt ©

P/16

$\Re(A, B)$ $S(B, C)$

$\Re \bowtie (\sigma_{B<5}(S))$

This may have unmatching tuples for $\Re$ since we are selecting tuples of S under condition $\sigma_{B<5}$.

But these unmatched tuples are not displayed.

a) $\sigma_{B<5}(\Re \bowtie S)$

↳ contains only matching tuples

↳ selection

Now this doesn't contain unmatched tuples.

b) $\sigma_{B<5}(\pi_{LOS}(S))$

↳ contains unmatche tuple

↳ unmatche tuple are removed

c) $\pi_{LOS}(\sigma_{B<5}(S))$

This may have unmatched tuples

∴ opt (c)

Sub Quries:

Nested query: It is a query within a query.

(Query1
(Query2))

Query1 → outer query

Query2 → inner query

→ The result of inner query is given to inner query.

Eg: Consider

Professor

SSN	Name
1	A
2	B
3	C

Teaches

SSN	Cid
1	9
1	10
2	10

Course

Cid	Cname
9	DB
10	DS

Q: find name of the professor who teaches the course with cid 9.

→ One way to write a query for this is using this which requires join of 2 tables.

> select name
from Professor P, Teaches T, ~~Course~~
where P.SSN = T.SSN and ~~T.cid = C.cid~~ T.cid = 9;

This method is inefficient for larger tables.

Here condition of where class is checked on 2×3 = 6 tuples.

→ If prof has 2000 tuples & teaches has 3000 tuples then join would result in 6000,000 tuples and select condition has to ~~app~~ checked on 60 lakh tuples.

→ Rather than this we can first select tuples with cid=9 from teaches and then perform join.

This can be done using nested queries.

```
>select name
 from professor
 where SSN in (select SSN
               from Teaches
               where cid = 9);
```

This is evaluated as first inner query

```
> select name
  from professor
  where SSN in ('1');
```

∴ o/p:

Name
A

→ Here condition is check only on 3000 tuples and hence this more efficient. So no of tuples involved = 3000 + 2000 = 5000 (with 2000 tuples)

21/11/20

Q: Find name of the professor who teaches the course 'Database'.

```
> Select name
  from professor
  where SSN in
    (select ssn
     from teaches
     where cid IN
       (select cid
        from course
        where cname = 'DB'));
```

O/P:

name
A

→ If the same query is written using join, then would require lot of space.

Q:

Student

Name	Marks
A	90
B	80
C	70
D	60

Find name of the student with highest marks

```
>select name
 from student
 where marks = (select max(marks)
                from student);
```

O/P:

name
A

→ Here inner query gives a set but since '=' operator is used the set is converted into value implicitly.

Q: find name of student with 2nd maximum marks

```
> select name
  from student
  where marks = (select max(marks)
                 from student
                 where markrs < > (select max (marks)
                                   from student));
```

To make above query simple we use limit clause.

Eg: find name of student with 3rd maximum marks

Syntax:

```
limit x offset y
```

It skips first y entries and returns next x entries

∴

```
> select name
  from student
  order by marks desc
  limit 1 offset 2;
```

o/p:

name
c

Correlated subquery:

~~(outer)~~ (outer
(inner))

for every tuple of outer query, inner query is executed once.
(i.e., every row of inner query)

→ Here both outer & inner are dependent on each other and hence the name correlated.

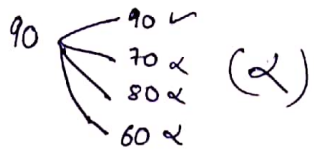
(row of outer query → Complete inner query → based on o/p of inner query, outer query is executed)

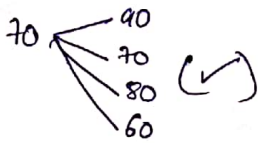
→ Now consider finding student name with third highest marks in below table

Student

Name	marks
A	90
B	70
C	80
D	60

> select s1.name
from student s1
where (select count(*)
from student s2
where s1.name =< s2.marks) = 3;





o/p. Name
B

However there is flaw with this query that it doesn't work with duplicate marks.

Eg:

Student

Name	marks
A	90
B	70
C	80
D	60
E	70

Our requirement is 3rd highest marks but not 3rd highest rank. So the expected o/p is

o/p:

Name
B
E

So the appropriate query can be written using DISTINCT

```
> select s1.name
  from student s1
  where ~~s1.marks~~ ( select count(distinct s2.marks)
                       from student s2
                       where s1.marks <= s2.marks ) = 3;
```

Note:

→ Syntactically correlated subquery and nested subquery looks similar.

So whenever inner query uses reference of outer query we say query is correlated subquery.

Q G-05 The relation book(title, price) contains the titles and prices of different books. Assuming that no two books have the same price, what does the following SQL query list?

```
select title
from book as B
where (select count(*)
       from book as T
       where T.price > B.price) < 5;
```

a) titles of 4 most expensive books

b) titles of the fifth most inexpensive book

c) titles of fifth most expensive book

d) titles of five most expensive books

Sol:

Here we print B.title

if there are 4 or less than four books with price greater than B.price.

If there are 4 books whose price is greater than B.price then B is 5th expensive book.

If there are 3 such books then B is 4th expensive book

2 --- 3rd

1 --- 2nd

0 --- 1st

∴ It give 5 most expensive books

Exists:

This is used to compare with an empty set.

i.e., to find whether inner query has returned any row or not.

exists (empty set) = false

exists (non empty set) = true

Eg: Consider

student

Rno	name
1	A
2	B

enroll

Rno	cno
1	98
2	99

Q: Find name of the student who enrolled a course with cno 98

```
>select name
 from student
 where EXISTS( select *
               from enroll
               where student.Rno = enroll.Rno
                 and cno=98 );
```

o/p of inner query

1 < (1, 98) ✓, (2, 99) ✗ } (1, 98)

2 < (1, 98) ✗, (2, 99) ✗ } —

∴ o/p:

NAME
A

Set Comparision Operators

Any / some :

$x < any(5, 10, 15)$

$\equiv (x<5)$ OR $(x<10)$ OR $(x<15)$

ALL :

$x < all(5, 10, 15)$

$\equiv (x<5)$ AND $(x<10)$ AND $(x<15)$

Eg: Consider

Instructor

Id	Name	dept	Salary
1	A	CS	50k
2	B	CS	60k
3	C	ME	90k
4	D	EC	55k
5	E	CE	40k

Q: Find name of the instructor ~~which~~ whose salary is greater than some instructor of the department CS

```
> selet a.name
  from instructor a
  where salary > Any (select salary
                      from instructor b
                      where b.dept = 'cs');
```

o/p:

Name
B
C
D

Similarly replacing 'any' with 'all' in the above query,

we have o/p:

Name
c

Note:

→ all (empty) is always true

$x \left\{ \begin{matrix} < \\ <= \\ = \\ > \\ >= \end{matrix} \right\}$ all (empty) is true.

→ any (empty set) is always false

$x \left\{ \begin{matrix} < \\ <= \\ = \\ > \\ >= \end{matrix} \right\}$ any (empty) is false.

All evaluates until it finds false and once it finds false it returns false. For empty set it doesn't find any false value so it returns true

→ x = any (s) ≅ x in (s)

Eg: select I_2.name
from instructor I_2
where dept <> 'cs'
and salary = all (select I_1.salary
from instructor
where dept = 'IT');

o/p: 3 rows

Eg: Select I_2.name
from instructor I_1
where dept <> 'cs'
and salary = any (select I_1.salary
from instructor
where dept = 'IT');

o/p: 0 rows

22/11/19

Multicolumn Subquery

Consider

Student(sid, ...)

Takes(sid, cid, sem, year)

1	99	I	2005
1	100	II	2006
2	99	I	2005
1	99	I	2006

Instructor(Id, ...)

Teaches(Id, cid, sem, year)

9	99	I	2005
10	98	II	2006
9	99	II	2006

Q: Find the total no of distinct students who have taken some course in some sem and year taught by an instructor with id = 9.

→ Here we need to compare multiple attributes.
i.e., pair wise comparsion.

→ we can do this using IN and this type of query is called multicolumn subquery

```
>select count(distinct sid)
 from takes
 where (cid, sem, year) in (select cid, sem, year
                            from teaches
                            where id = 9);
```

(cid, sem, year) → Pair wise Comparision

o/p:

count (distinct sid)
1
2

Unique:

The unique construct tests whether a subquery has any duplicate tuples in its result.

- The unique construct evaluates to "true" if a given subquery contains no duplicates otherwise false.

Eg. Consider

Course

Cid	Cname
9	DB
10	DS
11	PL

offerings

cid	sem	year
9	I	2009
9	II	2009
10	I	2009
11	II	2010

Q: find all courses that were offered atmost once in 2009.

```
select cname
from course
where unique ( select cid
               from offerings
               where course.cid = offerings.cid
                     and year = 2009);
```

o/p:

Cname
DS

With :

The 'with' clause provides a way of defining a temporary relation whose definition is available only to the query in which the with clause occurs.

→ 'with' clause helps write complex queries in a simple way.

Eg: Consider

Instructor

Id	Name	dept	salary
1	A	CS	20k
2	B	CS	20k
3	C	CS	30k
4	D	IT	30k
5	E	IT	30k
6	F	ME	20k

Q: Find all departments whose total salary is greater than average total salary at all departments.

total salary of CS = 70k

total salary of IT = 60k

total salary of ME = 20k

avg total salary of all depts $= \frac{70+60+20}{3} = 50k$

> with total_sal (dept, total) as
(select dept, sum(salary)
from instructor
group by dept)
dept_avg (total2) as
(select avg(total)
from total_sal)

with clause

total_sal

dept	total
CS	70k
IT	60k
ME	20k

dept_avg

total2
50k

Here total_sal, dept_avg are temporary relations created using with clause

select dept
from total_sal, dept_avg
where total > total2;

∴ o/p:

dept
CS
IT

→ The above query can also be written as shown below without using 'with' clause

> select dept
from instructor
group by dept
having sum(salary) >
(select sum(salary)/count(distinct dept)
from instructor)

Transactions & Concurrency Control

Transaction:

It is a collection of operations that perform a single logical unit of work.

Eg: Consider transfering Rs.100 from account A to B.

T: $A \xrightarrow{100} B$

Read(A)	A 500	B 1000
A = A - 100		
write(A)	A 400	B 1000
Read(B)		
B = B + 100		
write(B)	A 400	B 1100
Commit	→ values will be saved permanantly and rollback (undo) is no more possible.	

→ once transaction is commited it cannot be rolled back.

Properties of a transaction (ACID Properties):

i) Atomicity:

A transaction has to execute either all of the operations or none of the operations.

→ Transaction manager (component of DB) checks if this property is maintained or not.

(ii) Consistency:

→ State of system has to be correct always.

(For every transaction, we may have a consistency requirement)

Eg: In T: A $\xrightarrow{100}$ B

A+B = 1500 is a consistency requirement

→ This consistency is ensured by user or app. programmer.

(iii) Isolation:

Each transaction is unaware of other transaction that are executing.

i.e., Each transaction is executing independently.

→ Concurrency control manager supports this isolated execution of transactions.

(iv) Durability:

All updations done by a transaction should become permanant.

Schedule:

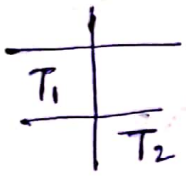
Schedule represents the order in which the operations of transactions ~~execution~~ execute. Schedule is of 2 types:

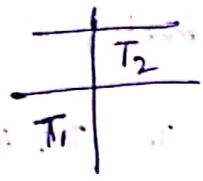
(i) Serial schedule

(ii) Concurrent schedule.

(i) Serial Schedule:

Each transaction is executed one after other without ~~inter~~ intervention.



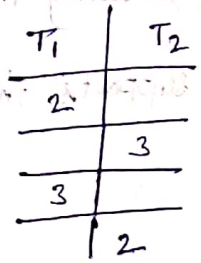


→ A serial schedule always ensures consistency.

(ii) Concurrent Schedule:

→ Here intervention is possible.

Let T_1 & T_2 be 2 transaction with 5 operations in each.



→ Concurrent schedule may lead to inconsistent state.

Eg:

T_1	T_2
R(A)	
A:= A+1000	R(A)
w(A)	A:= A+1000
	w(A)

} I/O & CPU operations can be done parallelly ∴ Concurrent schedule allows efficient resource utilization.

→ Concurrent schedules minimize waiting time, response time

Note:

→ no of serial schedules possible with n transactions = n!

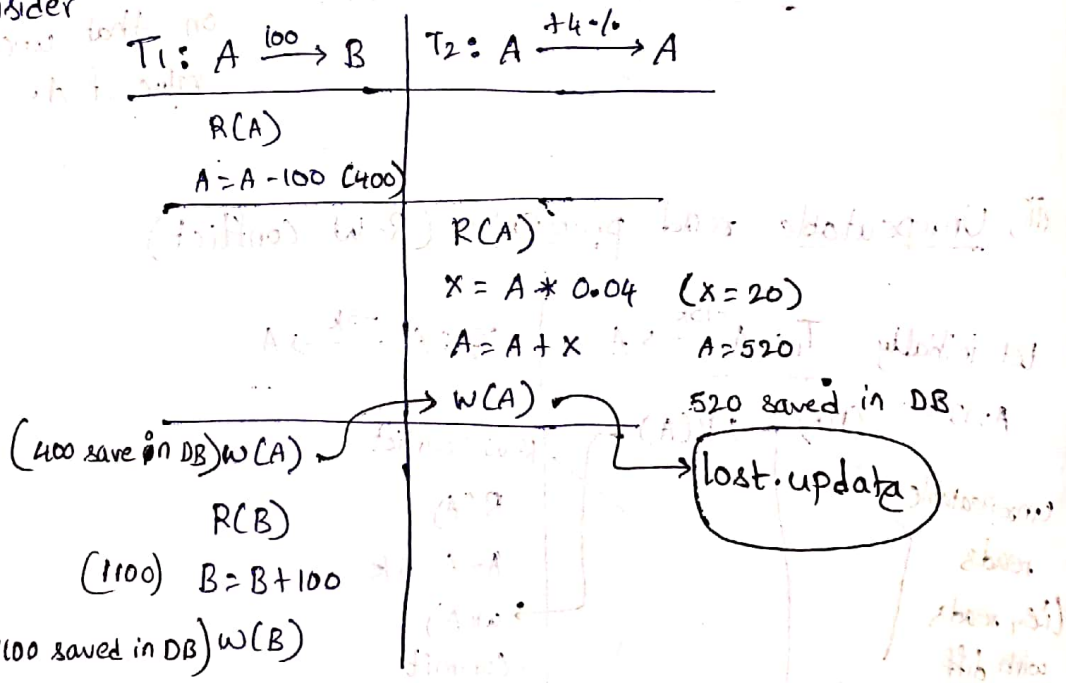
→ no of concurrent schedules possible with n transaction where each transaction has $m_1, m_2, \ldots m_n$ operation is

$$\frac{(m_1+m_2+\ldots+m_n)!}{m_1! * m_2! * \ldots * m_n!}$$

Problems due to concurrent schedule:

(i) Lost update problem (w-w conflict)

Consider



Assume initially A=500 & B=1000

After operation A = ~~600~~ 400 B=1100

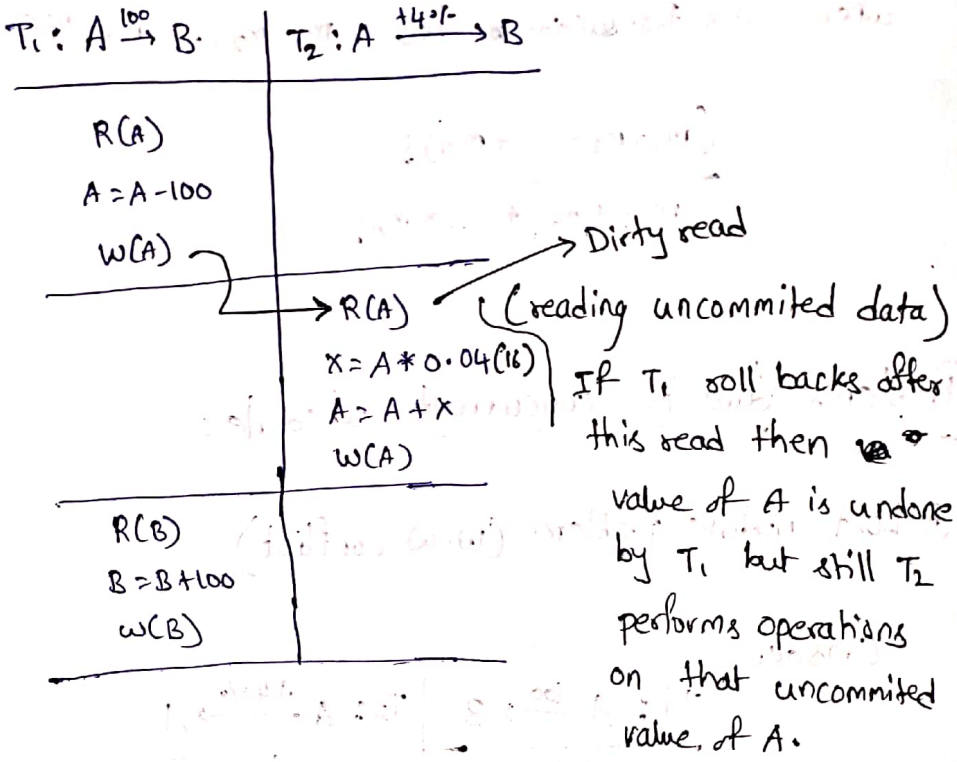
but correct state is A=420 & B=1100

Defn: If update of one transaction is overwritten by the update of another transaction without reading it, then it causes lost update problem.

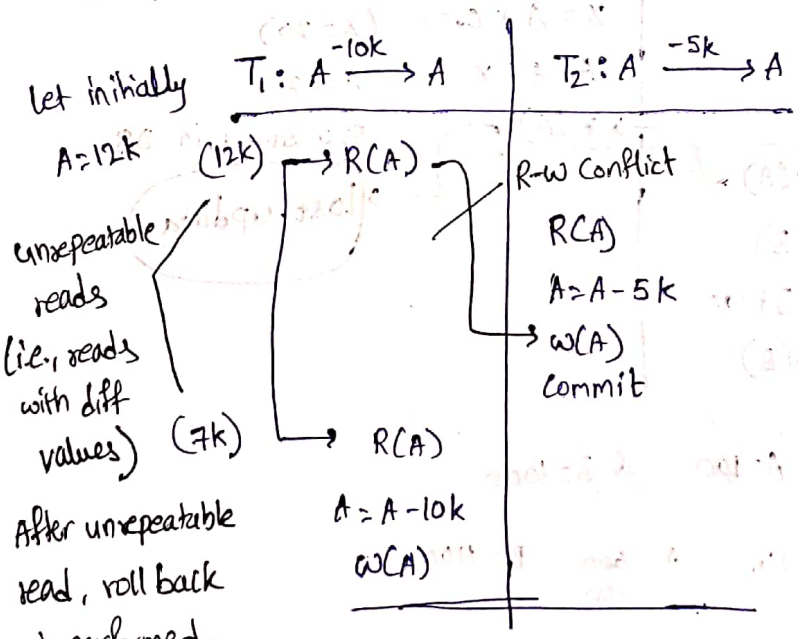
(ii) Dirty Read Problem (W-R conflict)

→ Reading uncommited data is called dirty read.

Consider



(iii) Unrepeatable read problem: (R-W conflict)



If we allow T_i to continue even after the unrepeatable read, assume a case where T_i fails and needs to roll back.

Since T_1 is depending on T_2, it also requires T_2 to roll back which is not possible cuz, T_2 has commited already.

∴ unrepeatable reads are not allowed.

Note:

Checking for above 3 problems in a given schedule.

(i) Lost update:

$w_i(A)$ followed by $w_j(A)$

T_j performs $w_j(A)$ without reading value produced by T_i from $w_i(A)$

(ii) Dirty read:

$w_i(A) \rightarrow R_j(A)$

$R_j(A)$ comes before commit of T_i

(iii) Unrepeatable read:

$R_i(A)$ —— $w_j(A)$ —— $R_i(A)$

(old value is read) (new value is read)

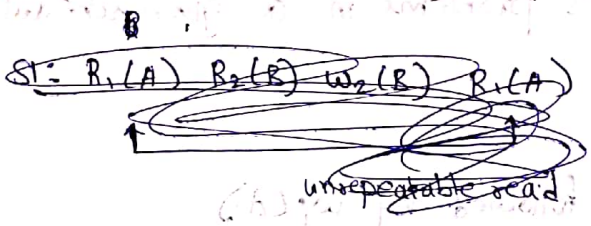
Eg: Consider schedules

S1: $R_1(A)$ $R_2(B)$ $W_2(B)$ $R_1(A)$ $W_1(A)$ $R_2(A)$ $W_2(A)$ C_1 C_2

S2: $R_1(A)$ $R_2(B)$ $W_2(B)$ $R_2(A)$ $W_2(A)$ C_2 $R_1(A)$ $W_1(A)$ C_1

S3: $R_1(A)$ $R_2(B)$ $W_2(B)$ $R_2(A)$ $W_1(A)$ $W_2(A)$ C_1 C_2

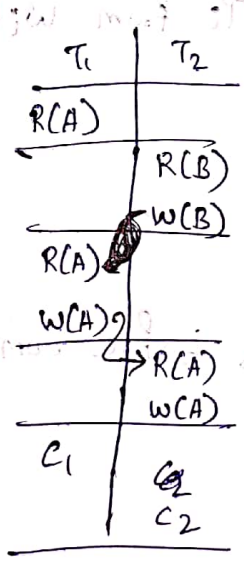
(i) which schedule has unrepeatable read?



Ans: S2

(ii) which schedule has dirty read problem?

S1:



∴ Ans: S1

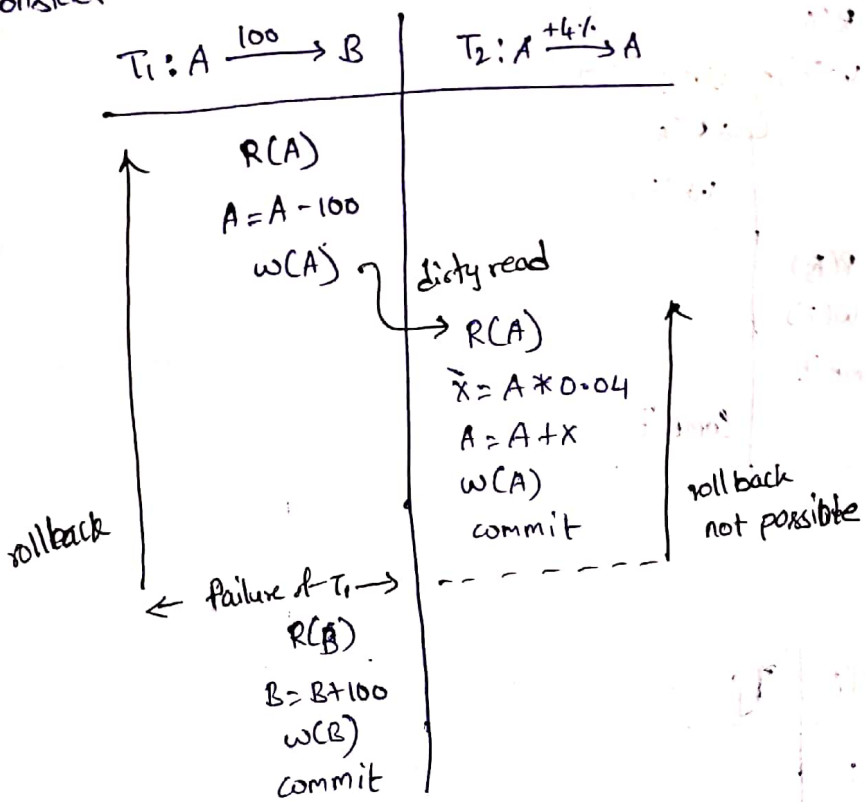
(iii) which schedule has lost update problem.

Ans: S3

Types of schedules based on recoverability

i) Non-Recoverable Schedule

Consider



→ Here operation of T_2 is dependent on T_1.

But T_2 has commited before T_1 and incase if T_1 needs to rollback then so should T_2 but this is not possible ~~because T_2 has~~

This is known as non-recoverable schedule.

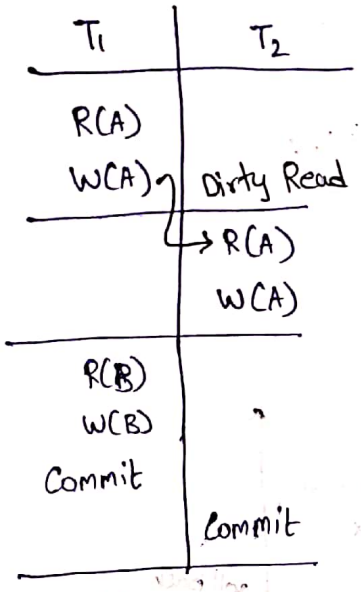
→ So the schedule will be recoverable if T_1 commits before T_2.

→ Non-recoverable ⇒ Dirty read exists.

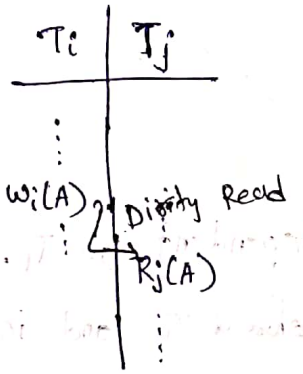
→ No Dirty read ⇒ Recoverable

(ii) Recoverable:

→ Same as previous case where T_2 commits after T_1



Note:



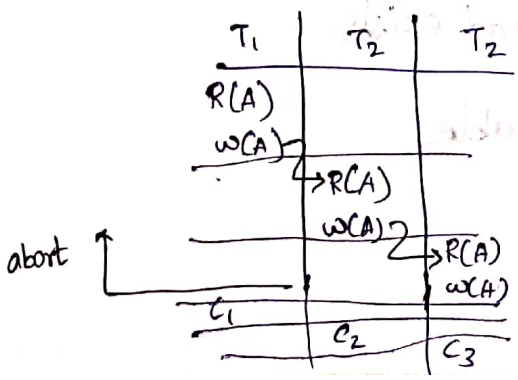
→ T_j commits before T_i ⇒ Non-Recoverable

→ T_j commits after T_i ⇒ Recoverable

24/11/20

(iii) Cascading Rollbacks / Cascading Aborts:

Consider below schedule



If T_1 needs to abort & rollback, it would cause T_2 to abort & rollback which further require abortion of T_3. This called as caseading rollbacks / caseading aborts.

Defn: A schedule is called cascading rollbacks, if abortion of one transaction leads to abortion of other transactions

(iv) Cascadeless Rollbacks:

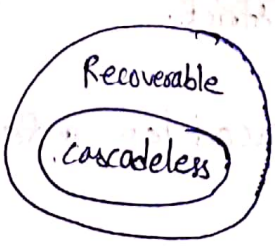
Consider below schedule,

T_1	T_2	
R(A)		
W(A)		
	R(B)	
	W(B)	
R(C)		
W(C)		
Commit		
	R(A)	⟶ Commited read
	W(A)	
	~~Commit~~ Commit	

Here failure of T_1 doesn't cause T_2 is abort.

So this schedule is called cascadeless rollbacks.

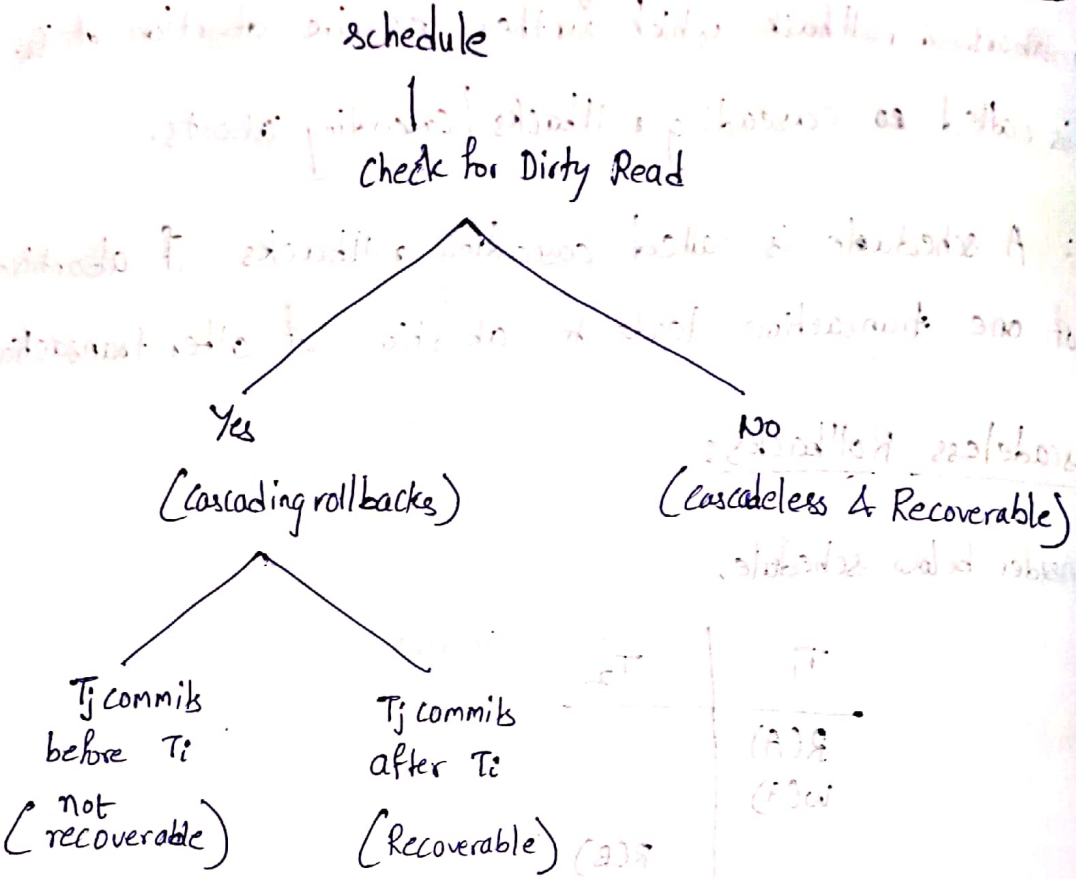
→ Every cascadeless schedule is recoverable but not viceversa



Cascadig rollback
⇒ Dirty read
no dirty ⇒ no CR

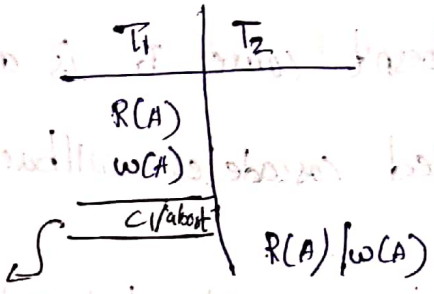
→ ~~Dirty Read ⇒ Cascading Rollbacks~~

Note:



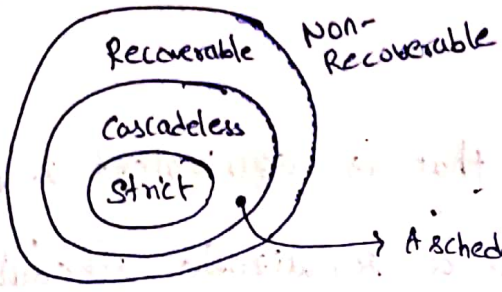
(v) Strict Schedule:

It is schedule which doesn't have dirty read & lost update problem. i.e. no ~~R-R~~ W-W, W-R, R-W conflits



This must be present for the schedule to be strict.

→ Strict schedule is cascadeless & Recoverable (∵ no W-R)



→ A schedule may be cascadeless & recoverable but not strict. This possible by ww conflicts

Eg:

T_1	T_2
R(A)	
w(A)	
	w(A)
C_1	
	C_2

(P/4) Serial → 2! = 2

Concurrent → $\frac{(3+3)!}{5!\,3!} = \frac{8 \times 7 \times 6}{3!} = 56$

non-serial → 56 − 2 = 54

(P/9) T_2 performed dirty read & commited first

∴ non-recoverable.

If it is not recoverable T_1 cannot be rolled back and only some part of T_1 is executed.

∴ not ensuring atomicity

Serializability:

Any concurrent schedule that is equivalent to some serial schedule is said to be a serializable schedule.

two types:

i) Conflict serializability

ii) View serializability

i) Conflict serializability:

Any concurrent schedule that is conflict equivalent to some serial schedule is said to be a conflict serializable schedule.

→ Two schedules are said to be conflict equivalent iff both the schedules have same set of conflicts.

Conflict operations:

$W_1(A)$ $W_2(A)$

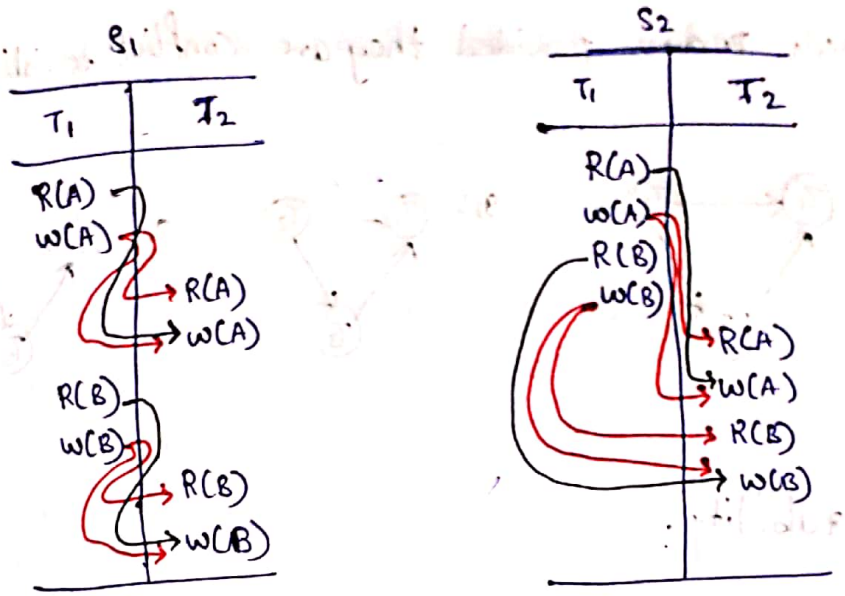
$W_1(A)$ $R_2(A)$

$R_1(A)$ $W_2(A)$

→ If two schedules have same set of conflicts then the result of both the schedules will be same.

So a conflict serializable schedule has equal results as that of a serial schedule which is consistent.

Eg: Consider two schedules



Conflicts in S_1 ≅ Conflicts in S_2

∴ S_1 & S_2 are conflict equivalent.

Since S_2 is ~~concurrent~~ serial schedule,

S_1 is said to be view serializable.

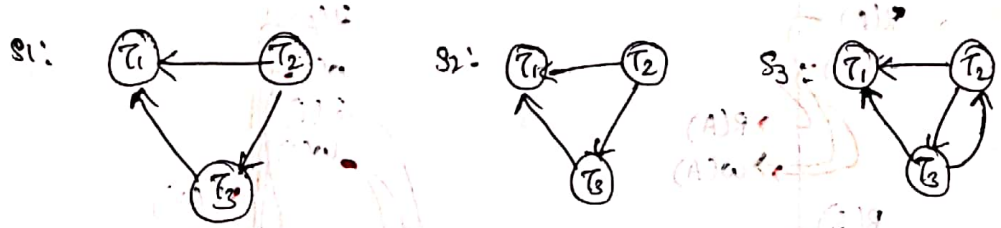
Test for conflict serializability:

(i) Construct a precedence graph in which each transaction is a vertex and each conflict operation from T_i to T_j is a directed edge from T_i to T_j

(ii) If the resultant graph contains no cycles, then the schedule is conflict serializable.

(iii) The order of serializability is determined based on topological sort of the precedence graph.

P/17 schedules with same precedence graphs are conflict equivalent ~~product~~ provided they are conflict serializable.

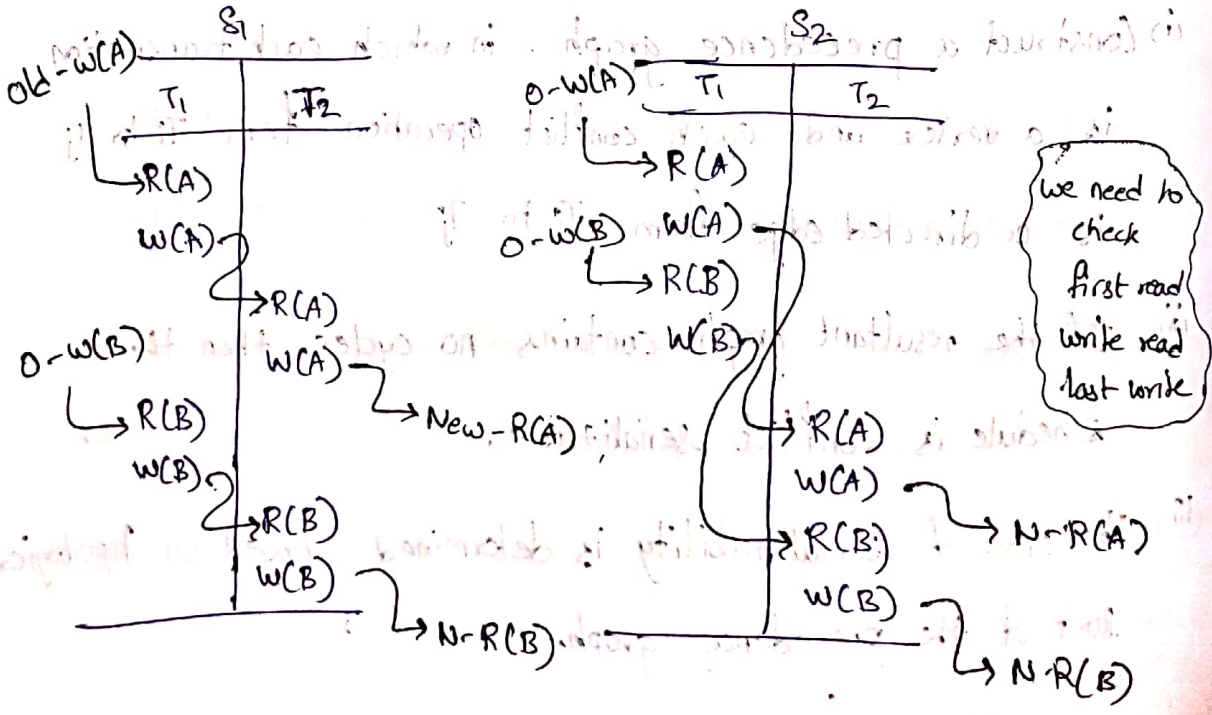


View Serializability:

→ A concurrent schedule is said to be view serializable if it is view ~~equa~~ equivalent to some serial ~~sch~~ schedule.

→ Two schedules S1 & S2 are said to be view equivalent iff all the views $(W_i(Q) - R_j(Q))$ are same in both the schedules.

Eg: Consider below schedules

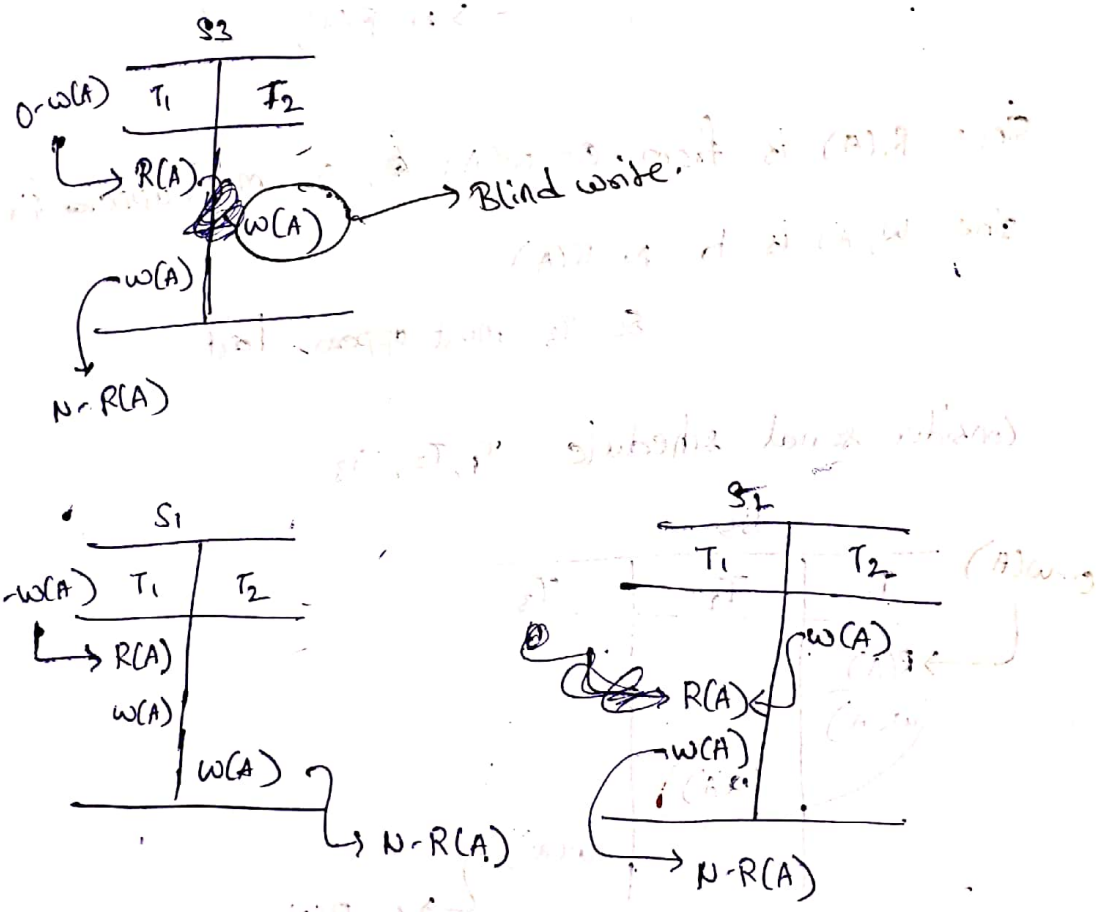


→ S_1 & S_2 has same views

∴ S_1 is view equivalent to S_2 and S_2 is a serial schedule

∴ S_1 is view serializable.

Eg: Check whether below schedule is view serializable or not.



∴ S_3 is neither equivalent to S_1 nor to S_2

∴ S_3 is not view serializable.

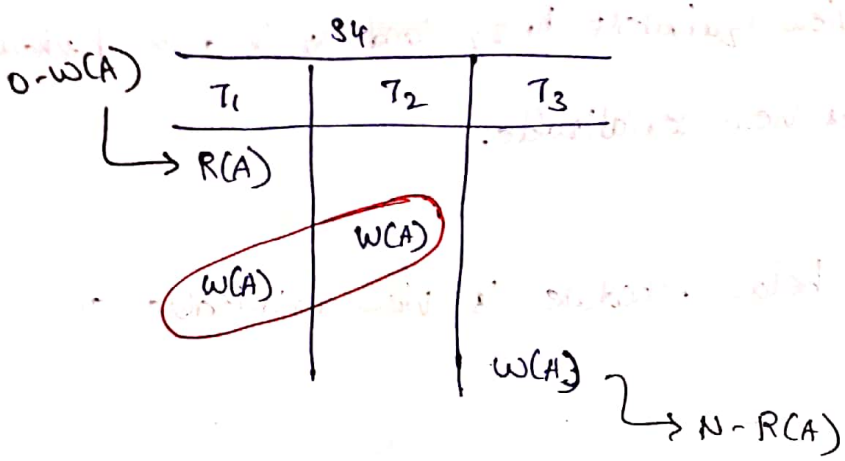
Note:

→ Testing view serializability is ~~called~~ NP hard problem.

Blind write:

A write performed without read is called blind write.

Eg: Check whether below schedule is view serializable or not.

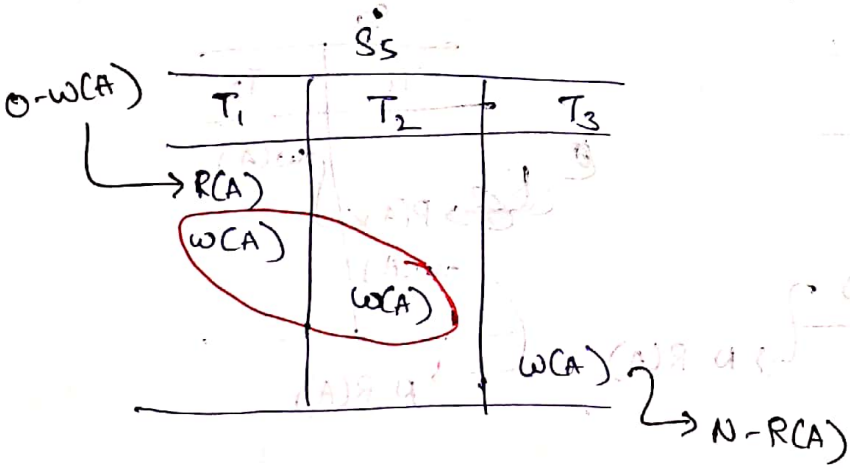


Since $R_1(A)$ is from 0-W(A), T_1 must appear first

Since $W_3(A)$ is to N-R(A)

So T_3 must appear last

Consider serial schedule T_1, T_2, T_3



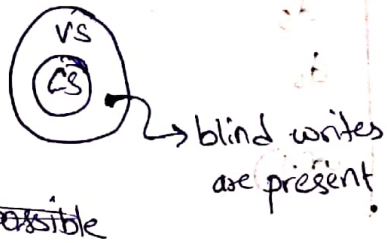
The order of write operations in S_4 & S_5 (encircled) are different but still S_4 is view equivalent to S_5.

S_4 is view serializable but not conflict serializable.

∴ view serializability is not as powerful as conflict serializability.

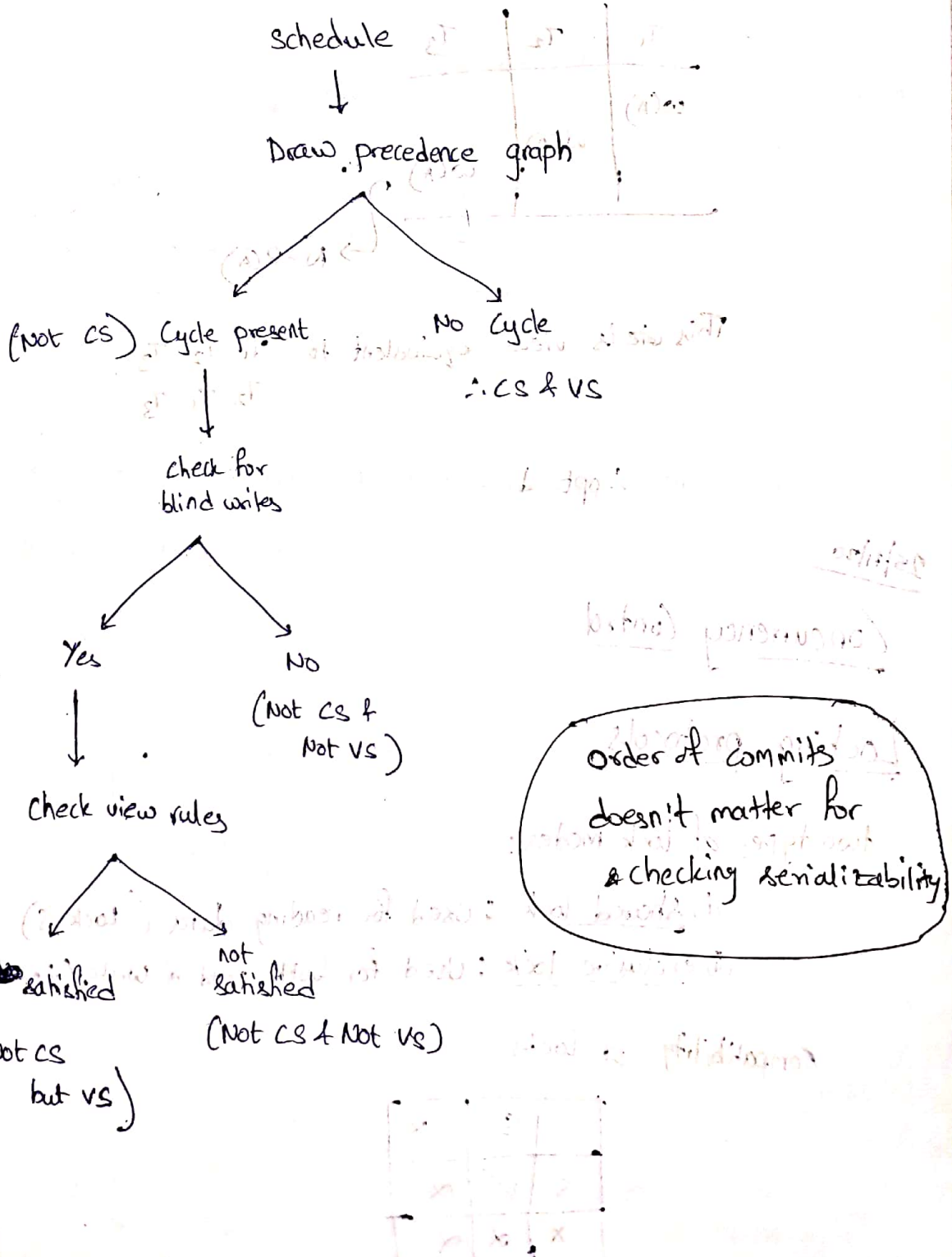
Note:

→ Every conflict serializable schedule is also view serializable



~~→ Blind writes are possible~~

→ A schedule is VS but not CS ⇒ Blind writes exist.



(P/19)

T_1	T_2	T_3
d_1	d_3	d_5
d_2	d_4	d_6
w(A)	w(A)	w(A)

Here only order of w(A) matters

Consider

T_1	T_2	T_3
w(A)		
	w(A)	
		w(A)

→ N-R(A)

This ~~use~~ is view equivalent to $T_1\ T_2\ T_3$
$T_2\ T_1\ T_3$

∴ opt d

25/11/20

Concurrency Control

Locking protocols

two types of lock modes:

i) shared lock : used for reading data (lock-S)
ii) exclusive lock : used for both read & write (Lock-X)

Compatibility of locks

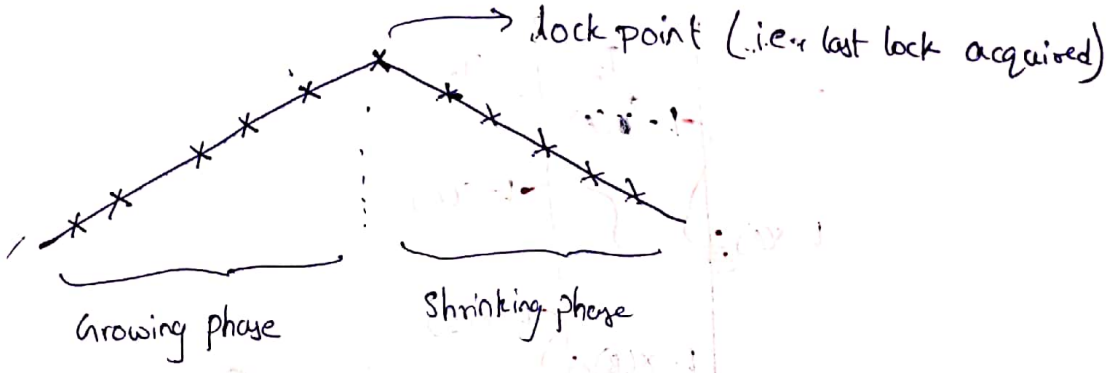
	S	X
S	✓	✗
X	✗	✗

Two-phase locking protocol (2PL):

Locking and unlocking is done in 2 phases

(i) Growing phase: Transactions can acquire the lock but can't release the locks.

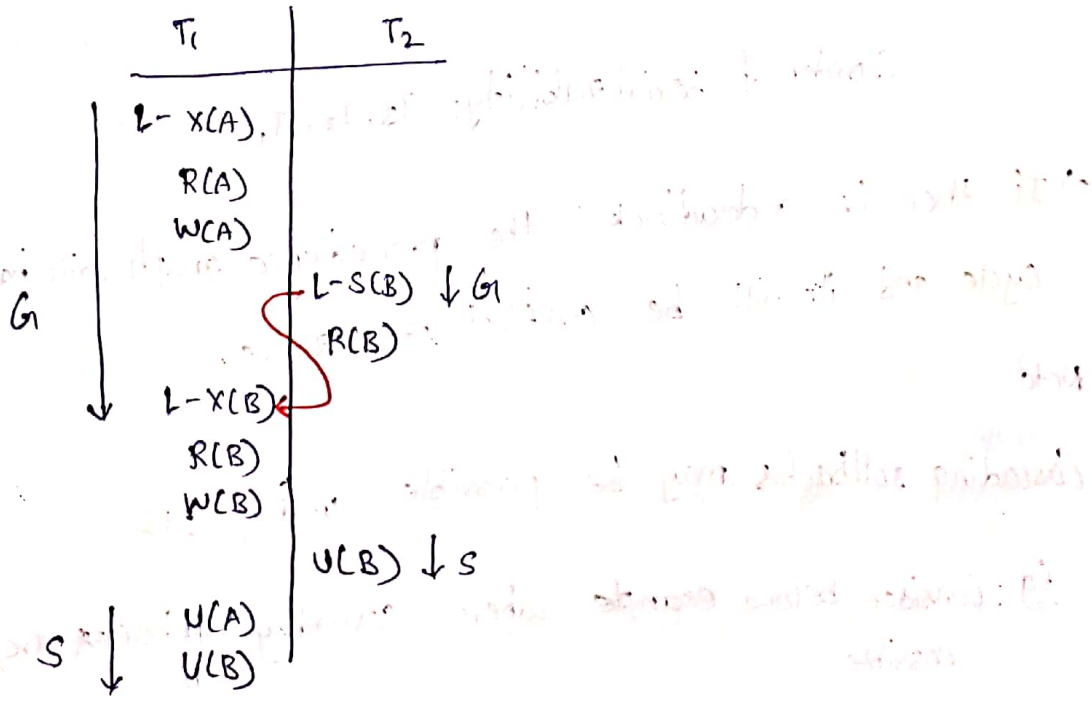
(ii) shrinking phase: Transactions can release the locks but can't obtain the locks.



Note:

Every schedule possible under 2PL always ensures conflict serializability. (if no deadlocks)

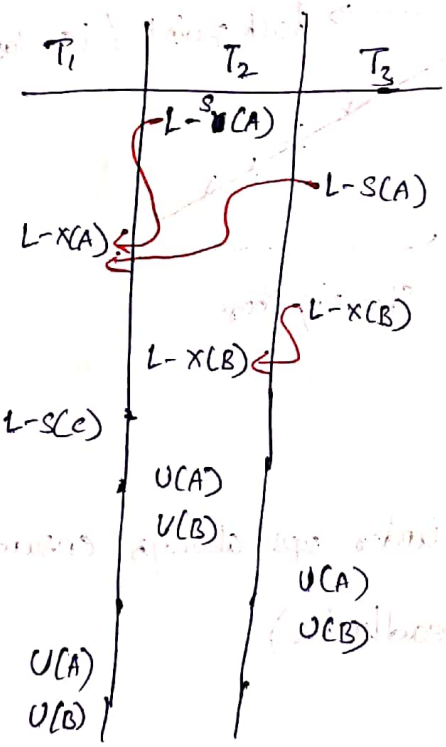
Eg: Consider

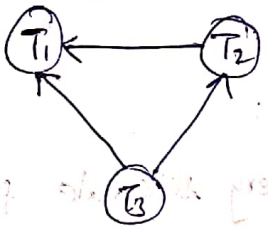


→ Here we draw predcedence graph where edges represent conflicting locks.

∴ order of serializability: T_2, T_1

Eg: find the serializability order of the following serializable 2PL schedule





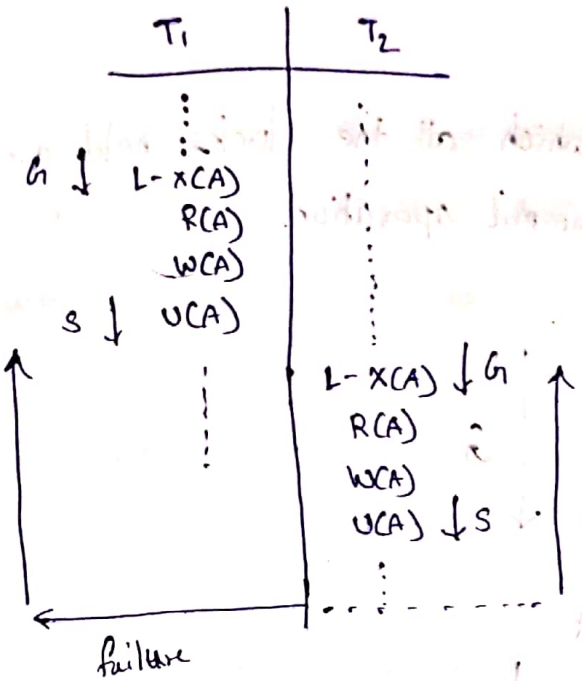
∴ order of serializability: T_3, T_2, T_1

→ If there is a deadlock, the precedence graph will have cycle and it will be conflict serializable.

Note:

Cascading rollbacks may be possible under 2PL.

Eg: Consider below example where cascading rollbacks are possible



If T_1 fails, T_2 should also roll back since there is a dirty read.

→ To avoid cascading rollbacks there are 2 modifications of 2PL

(i) Strict 2PL

(ii) Rigorous 2PL

(i) strict 2PL:

It is 2PL in which all the exclusive locks held are released only after commit.

(Now check above example with this modification)

Eg:

	T_1
G	L-X(A)
	R(A)
	W(A)
	L-S(B)
	R(B)
	U(B)
S	commit
	U(A) → exclusive lock released after commit

Since exclusive lock is released after commit, there will be no chance of W-R & W-W conflicts as in the case of strict schedule

∴ it is strict 2PL

(ii) Rigorous 2PL:

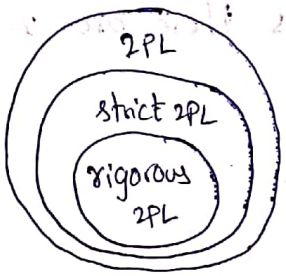
It is a 2PL in which all the locks held are released only after commit operation.

Eg:

T_1	
L-X(A)	↓ G
R(A)	
W(A)	
L-S(B)	
R(B)	
Commit	
U(A)	↓ S
U(B)	

Note:

→ Every rigorous 2PL is strict 2PL



Note:

Deadlocks are possible under 2PL

T_1	T_2
L-X(A)	
	L-X(B)
L-S(B)	
	L-S(A)
Commit	
	Commit
U(A)	
U(B).	
	U(A)
	U(B)

T_1 under waiting state

T_2 under waiting state

∴ Deadlock.

Note:

→ Basic 2PL ensures conflict serializability.

→ Strict & Rigorous 2PL ensures conflict serializability, ~~rollback~~ ~~cate~~ cascadeless schedure, recoverable.

Deadlock Handling:

(i) Deadlock Prevention:

It ensures that deadlock never occurs.

There are 2 approaches for deadlock prevention.

a) Wait-Die: older waits, younger dies.

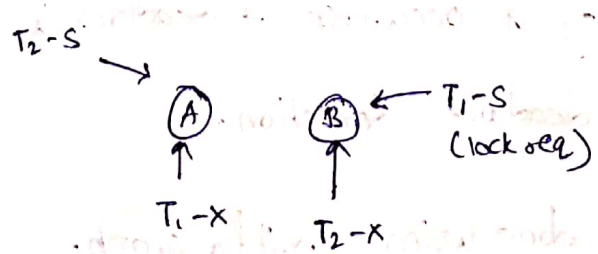
Every Tx has a timestamp. Older Txs have lower time stamps.

Eg: $TS(T_i) < TS(T_j)$

$\Rightarrow T_i$ is older process.

Eg: Consider

(older) T_1	(younger) T_2
L-X(A)	
	L-X(B)
L-S(B)	
	L-S(A)



Here since T_1 is older, T_1 waits.

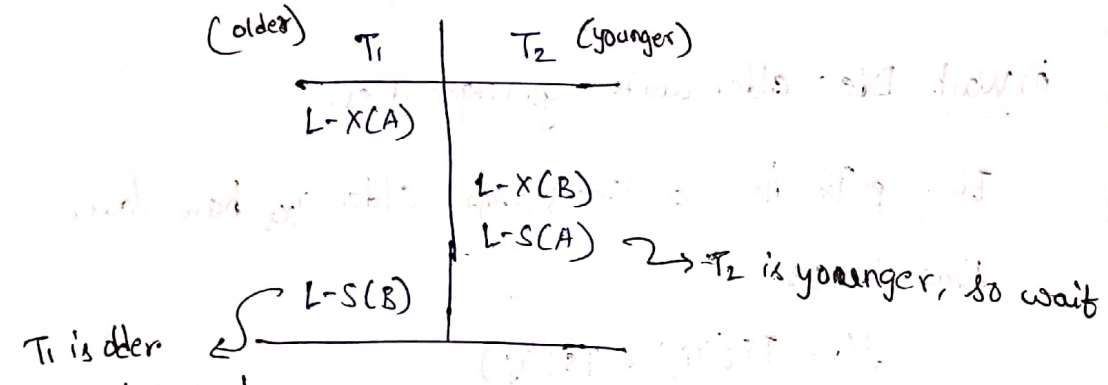
Later when T_2 request lock on T_1, T_2 must die since T_2 is younger than T_1.

So T_2 is aborted & rolled back and restarted with same time stamp. (To avoid starvation.)

Once T_2 dies T_1's wait is over and continues with execution.

b) ~~Word~~ wound-wait: older wounds, younger waits.

Eg: Consider



27/11/20

(ii) Deadlock detection:

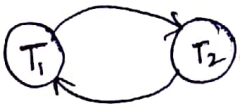
Sometimes deadlock prevention causes roll backs even if there is no deadlock.

So when probability of occurence of deadlock is less, it is better to go for deadlock detection.

→ This detection is done using wait for graph.

Eg:

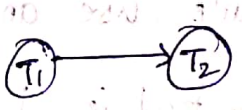
T_1	T_2
L-X(A)	
	L-X(B)
L-S(B)	
	L-S(A)
	U(B)
	U(A)
U(A)	
U(B)	



Cycle is present

∴ Deadlock has occured

T_1	T_2
L-X(A)	
	L-X(B)
L-S(B)	
	U(B)
U(A)	
U(B)	



No cycle

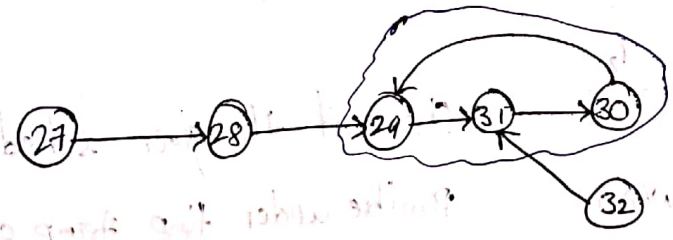
∴ No deadlock.

→ Edge from T_1 to T_2 indicates T_1 is waiting for a resource held by T_2.

→ In 2nd example after U(B) is executed by T_2, the edge from T_1 to T_2 is deleted.

→ When the system detects deadlock, the system selects a transaction from the cycle and aborts it.

P/24



∴ Deadlock is occured.

Time stamp based protocols:

Here each T_x is given a unique timestamp.

T_x with less timestamp value are more older.

→ This protocol say, every schedule must be equivalent to serial schedule T_i, T_j such that

$$TS(T_i) < TS(T_j)$$

→ This doesn't use any locks.

∴ This protocol is deadlock free protocol.

→ Here the order of serializability is in the order of the timestamps.

❖ 2 types of time stamp based protocols:

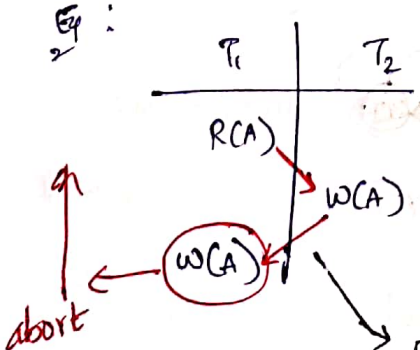
(i) Time stamp ordering protocol.

(ii) Thomas write rule.

(i) Time stamp ordering protocol:

Here all the conflicting operations must be excuted in the order of their time stamps.

Eg:



Check if the given schedule is possible under time stamp ordering protocol.

Sol:

This conflict doesn't follow the required order of serializability. So abort T_1.

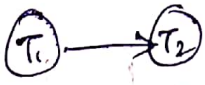
To fall under time stamp ordering protocol it must be conflict equivalent to serial schedule T_1, T_2

But he the precedence graph is

∴ This schedule is not possible under time stamp ordering protocol.

Eg: Is the below schedule possible under time stamp ordering protocol.

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



$TS(T_1) < TS(T_2)$

order of serializability: $T_1; T_2$.

∴ possible under TSOP.

Note:

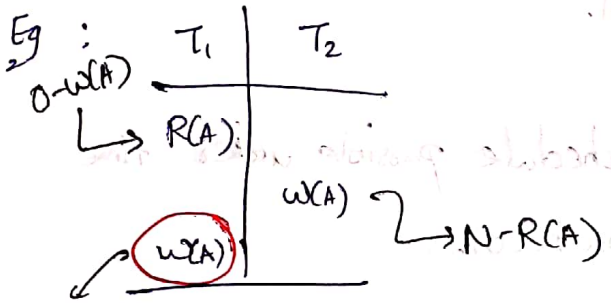
→ TOP ensures CS.
→ It ensures freedom from deadlock. } adv
→ Aborting & restarting of T_P (with new TS) may cause starvation } disadv

(ii) Thomas write rule:

→ It is a modification of TOP that minimizes the problem of starvation by avoiding the obsolete writes.

↳ useless write

Eg:



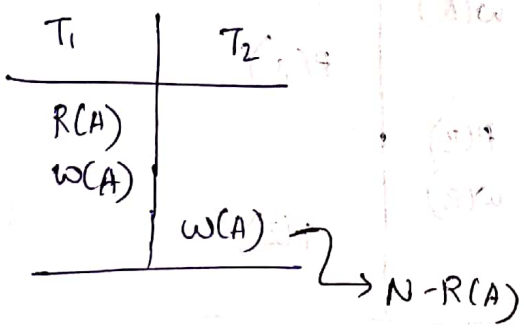
obsolete write.

$TS(T_1) < TS(T_2)$

∴ This should be equivalent to serial schedule $T_1 T_2$

∵ expected & needed $w_2(A)$ is already performed.

So we ignore $w_1(A)$.



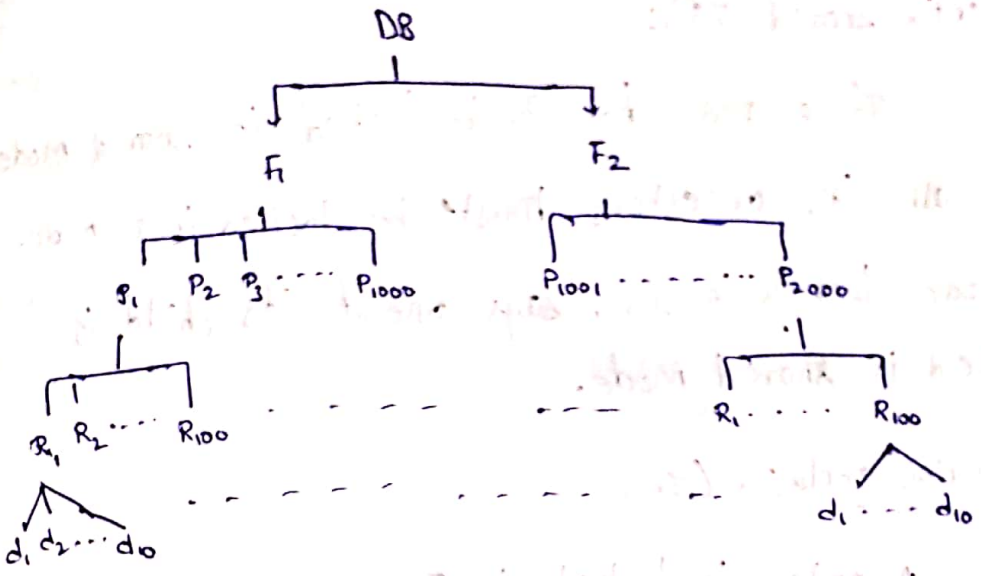
→ So the above schedule is allowed under Thomas write rule.

→ T.W.R ensures view serializability. by ignoring obsolete write.

Note:

→ Every schedule that is possible under TOP is possible under TWR

Multiple Granularity locking protocol:



→ If a lock is need on all 10 data items of a record, a lock is requested on record rather than each data item individually and thus it reduces no of locks to be requested.

→ If P_1 is locked in shared mode, it means all the descendants of P_1 are locked in shared mode.

~~Eg: Consider 2 transactions~~

~~T1: Read (R_1) of P_1~~

~~T2: update (F_1)~~

→ If a child node is locked, then its ~~parent~~ ancestor cannot be locked in conflicting mode. and vice-versa.

This system uses 3 additional lock modes

(i) Intention shared (IS):

If a node has to be locked in shared mode, then all its ancestors must be locked in IS mode. A node locked in IS, says one of its child is locked in shared mode.

(ii) Intention exclusive (IX)

A node is locked in IX says one of its child node is locked in exclusive mode.

(iii) Shared and Intention Exclusive (SIX):

A node locked in SIX says that the node is locked in shared mode and one of its descendants is locked in exclusive mode.

Eg: Consider two transactions

T_1: Read of R_1 of P_1

IS(DB)
IS(F_1)
IS(P_1)
S(R_1)
U(R_1)
U(P_1)
U(F_1)
U(DB)

T_2: Update(F_1)

IX(DB)
X(F_1)
U(F_1)
U(DB)

Multiple granularity 2PL (ensures cs)

→ Multiple granularity 2PL ensure CS.

→ locking is done from root to leaf.

→ Unlocking is done from leaf to root.

→ Here deadlock may occur.

28/11/20

File Organization & Indexing

File Organization:

Data Base

↓

set of Files

↓

set of blocks

↓

set of records

Eg: Consider a file with 1000 records and size of each record is 25 bytes. let block size be 100 bytes.

$$\text{no of records per block} = \frac{100}{25} = 4$$

$$\text{no of blocks require} = \frac{1000}{4} = 250$$

Blocking factor: It is the avg no of records that can be stored in a block.

$$Bf = \frac{\text{block size}}{\text{record size}}$$

Eg: Consider

Block size = 100 bytes; R = 30 bytes;

$Bf = \frac{100}{30} = 3.33$ records/blocks. (spanned strategy).

(or)

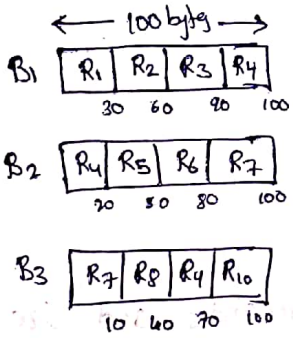
= 3 records/block (unspanned strategy)

→ For storing records into memory, we have 2 strategies

(i) Spanned strategy:

A part of record is possible to store in a block.

i.e., a record may span accross more than one blocks



Adv: No wastage of memory

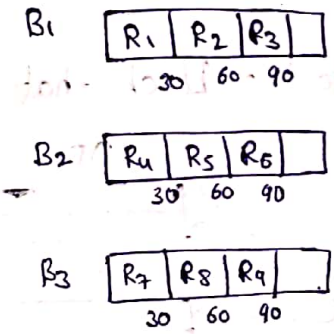
Disadv: More no of block accesses may be needed for one record

Block access: Moving block of records from disk to memory.

→ This technique is suitable for variable length records.

(ii) Unspanned strategy

→ No record is stored across multiple blocks.



adv : Less no of block accesses

Disadv : Wastage of memory.

→ Suitable for fixed length records.

→ For storing records into a file, we have two strategies

(i) Ordered file organization

(ii) Unordered file organization

Ordered file organization	Unordered file organization
→ Records are stored in the order of some key value	→ Any record can be stored anywhere in the file
→ Use binary search	→ Uses linear search
→ Searching is efficient	→ Storing records is easy.
→ Storing records is difficult	→ Searching records is difficult.

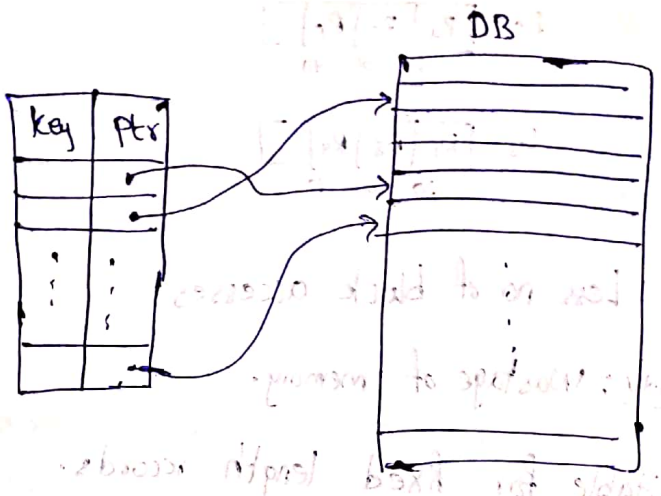
→ Most databases use ordered for pk but searching based on other fields is not efficient.

Index: Index is used to speed up the searching process.

→ Index is an ordered file with 2 fields (key & pointer)

~~key ptr~~

Here pointer points ~~the~~ to the block that contains the record.



Index is classified into 2 types:

(i) Single level index → Primary Index (Pk + ordered)
→ Clustered Index (NK + ordered)
→ Secondary Index (NK/CK + unordered)

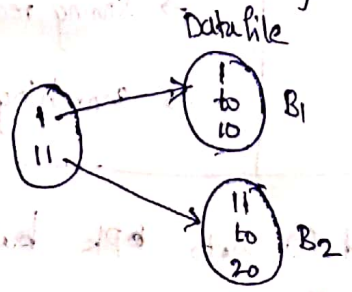
(ii) Multilevel index → B-Tree
→ B^+-Tree

Other classification of index:

(i) Sparse index:

~~Dense index~~

Index record is created only for some key values.



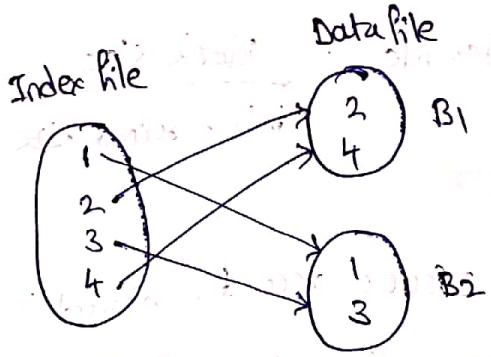
→ This is possible when data file is ordered.

→ no of index records = no of blocks.

→ Here pointer used is block pointer.

(ii) Dense index:

Index record is created for each record of datafile.



→ This is used when datafile is unordered.

→ no of index records = no of records in data file.

→ Here pointer used is record pointer.

Single level index:

(i) Primary Index: (Pk + ordered)

→ Primary index is created on pk if the file is ordered w.r.to Pk.

→ Type of index: sparse index

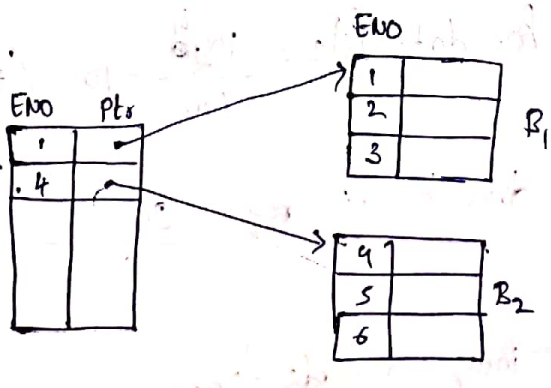
With indexing

→ Bi - no of index blocks

→ Binary search

→ avg no of block accesses

$= \log_2 B_i + 1$



Without indexing

→ B - no of data blocks

→ Binary search

→ avg no of block accesses

$= \log_2 B$

(i) Index record is created for each block (sparse index).

(ii) no of records = no of blocks in datafile

$$\text{BF for data file} = \frac{\text{Block size}}{\text{data record size}}$$

$$\text{BF for index file} = \frac{\text{Block size}}{\text{index record size}}$$

(Q1)

Eg: Consider a file of 30000 records stored in a datafile in the blocks where size of each block is 1024 bytes. Size of each record is 100 bytes of which key field is of size 9 bytes and both block and record pointer takes 6 bytes. If we create primary index on the key field of the file, then find the average no of block accesses to access a record.

Sol:

$n = 30000$; $B = 1000$ bytes; $R = 100$ bytes;

$k = 9$ bytes; $B_{ptr} = R_{ptr} = 6$ bytes

$$\text{BF for data file} = \frac{1024}{100} = 10 \quad (\because \text{fixed length records})$$

$$\text{no of data blocks req} = \frac{30000}{10} = 3000$$

$\Rightarrow$ no of index records = 3000

size of each index record = 9 + 6 = 15 bytes

Blocking factor for index file = $\left\lfloor \frac{1024}{15} \right\rfloor$ = 68

(∵ fixed length record)

⇒ no of index blocks req

$= \left\lceil \frac{3000}{68} \right\rceil$ = 45

∴ avg no of block accesses = $\lceil \log_2 45 \rceil + 1$

= 7

avg no of block accesses without index

= $\lceil \log_2 3000 \rceil$ = 12

Clustered index: (Nk + ordered)

→ Clustered index is created on non-key (duplicate values are possible) when data file is ordered w.r.to Nk

→ All the records with same Nk value forms a cluster.

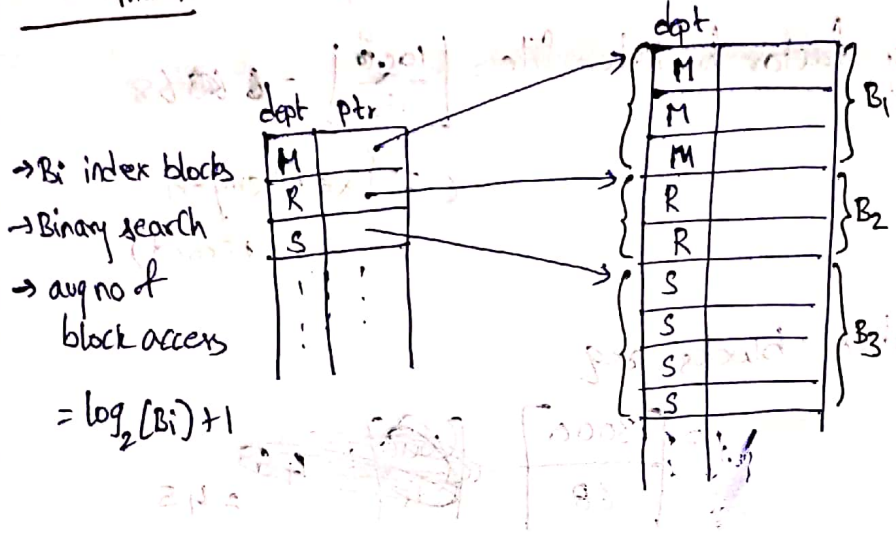
→ Each cluster is assumed as one block.

(i) Index record is created for 1st record of each cluster.

(ii) No of index records = no of clusters

∴ Sparse index.

with index



→ B_i index blocks
→ Binary search
→ avg no of block access
$= \log_2(B_i) + 1$

without index

dept

→ B - data blocks
→ Binary search
→ Avg no of block access
$= \log_2(B)$

Eg: (Q2) Suppose if we create a clustered index on data file of (Q1) then find the avg no of block accesses. no of clusters in the data file is 1000. Assume size of blk is 9 bytes.

Sol:

no of index records = 1000

BF for index file $= \left\lfloor \frac{1024}{615} \right\rfloor = 68$

no of index blocks $= \left\lceil \frac{1000}{68} \right\rceil = 15$

The avg no of block accesses $= \log_2 15 + 1$

$= 5$

without indexing,

no of block accesses $= \log_2(1000) = 10$

Note:

→ Data block size is variable (depending on cluster size)
→ Index block size is fixed.

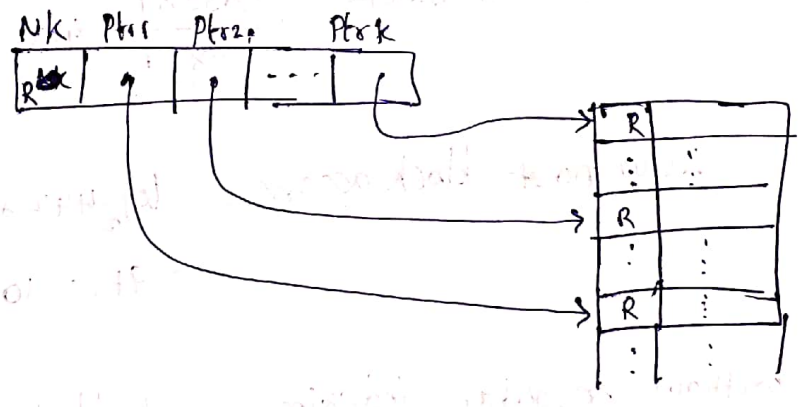
Secondary index (NK/CK + unordered)

→ Secondary index is created on NK/CK when PK is already present and the file is unordered w.r.to NK/CK.

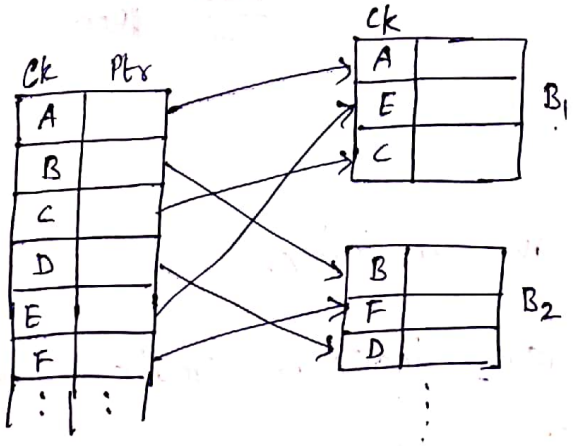
→ Secondary index is possible on NK, but it is not recommended.

It is because an index record needs to have many pointers and in some cases size of index record could be larger than data record.

Eg:



→ Thus it is recommended to have secondary index on CK.



→ Bi - index blocks

→ Binary search

→ avg no of block accesses

$= \log_2 Bi + 1$

→ B - data blocks

→ Linear search

→ Avg no of block access

$= B/2$

(i) Index record is created for each record (dense index)

(ii) no of index records = no of records in data file

Eg: Suppose if we create a secondary index on the datafile, then find the avg no of block accesses (Q3)

Sol:

no of index records = 30000

size of each index record = 15

no of index blocks = $\left\lceil \frac{30000}{68} \right\rceil = 442$

∴ avg no of block accesses = $\log_2 442 + 1$

$= 9 + 1 = 10$

without secondary indexing, no of block accesses

$= \frac{\text{no of data blocks}}{2}$

$= \frac{3000}{2} = 1500$

29/11/20

Multi-level Index:

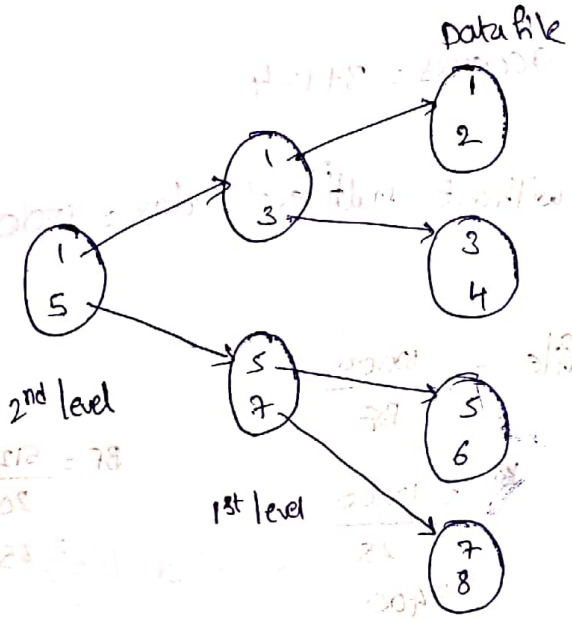
→ It is an index over an index.

→ On any 1st level index we create a primary index (∵ Index file is always ordered without any duplicates)

→ we keep on creating multiple indices until the last level index file fits in one block.

→ Here no of index records in n^{th} level = no of index blocks in $(n-1)^{th}$ level $(n>1)$

→ The avg no of block accesses = no of levels + 1



→ 1st level may be a sparse index or a dense index.
2nd and higher levels are always sparse indices.

Eg:
Q4) Suppose if we convert the secondary index created in the Q3) into multilevel index & then find the no of block accesses

Sol:

no of index blocks = 442

⇒ no of index records in 2nd level = 442

~~size of 2nd level index = 442 * 15~~

no of index blocks in 2nd level = $\lceil \frac{442}{68} \rceil = 7$

⇒ no of records in 3rd level = 7

⇒ no. of index blocks in 3rd level = $\lceil \frac{7}{68} \rceil = 1$

∴ 3-level index.

∴ avg no of block accesses = 3+1=4

No of block accesses without multilevel index = 1500

P/2

no of blocks in data file = $\frac{10000}{BF}$

$= \frac{10000}{25}$

$= 400$

$BF = \frac{512}{20}$

$= 25$

⇒ no of index records = 400

P/3

$r = 16384$; $R = 32$ byts; $k = 6$ byts;

ordered on Nk; Unspanned;

block size = 1024; BP = 10byts.

Secondary index:

$BF_{(Data\ file)} = \frac{1024}{32} = 32$

no of blocks in datafile = $\frac{16384}{32} = 512$

no of records in index file = no of records in data file

= 16384

Size of index record = 10 + 6 = 16

BF of index file = $\frac{1024}{16} = 64$

no of index blocks = $\frac{16384}{64} = 2^8 = 256$

no of records in 2nd level index = 256

no of blocks in 2nd level = $\frac{256}{64} = 4$

Implementation of Multilevel index:

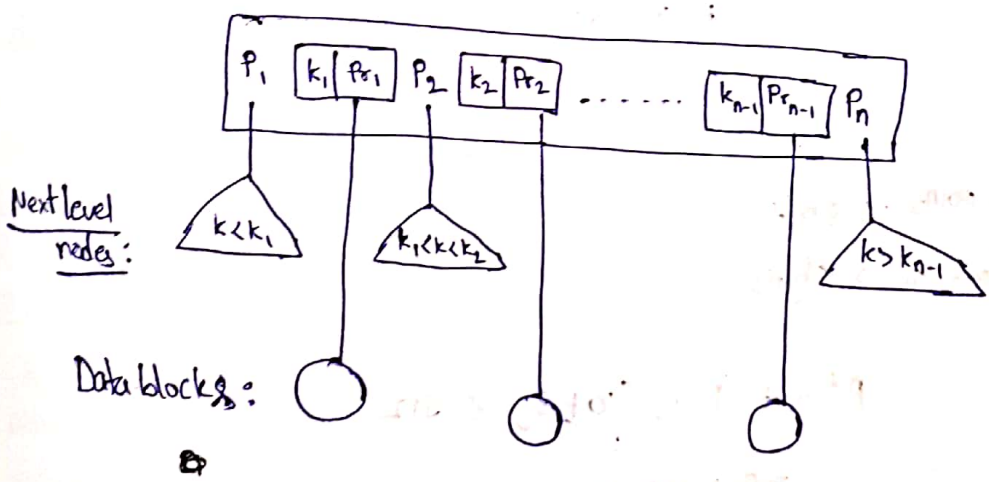
→ Multilevel index can be implemented using B-Tree & B^+-Tree.

B-Tree:

→ It is a height balanced search tree.

→ Each block in the multilevel index is assumed as one node in B-Tree

→ Structure of B-Tree node:



B-Tree provides direct access (∵ If key found at some level, we can directly move to data block)

In the structure of node

$P_1, P_2, \ldots P_n \rightarrow$ Block pointers

$Pr_1, Pr_2, \ldots Pr_{n-1} \rightarrow$ record / data pointer

$k_1, k_2, \ldots k_{n-1} \rightarrow$ keys

Here order of B-Tree = n

Order of B-Tree:

→ Order is max no of tree pointers in a node.

If n is order,

no of tree pointers = n

no of index records = n-1

(key + record ptrs)

If B is size of block

$$n*P + (n-1)*(k+P_r) <= B$$

P → size of block Ptr

P_r → size of record ptr

k → size of k

n → order

(P/4)

data ~~block~~ points → 8 byts

block ptr → 5 bytes

$$P*5 + (P-1)(10+8) \leq 512$$

$$5P + 18P - 18 \leq 512$$

$$23P \leq 530 \Rightarrow P \leq 23.04$$

$$\therefore P = 23$$

~~Insertion~~:

note:

A B-Tree of order n must have a minimum of

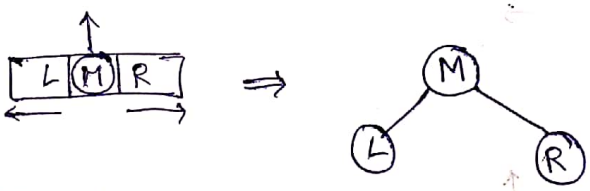
$\lceil \frac{n}{2} \rceil$ pointers

$\lceil \frac{n}{2} \rceil - 1$ keys

But requirement need not to be satisfied by root node.

Insertion:

→ If a node is full then split the node into two nodes and all the key values less than middle key move to left subtree and greater than middle key move to right subtree and middle node to upper level.



Eg: Insert following keys in a B-Tree of order 3

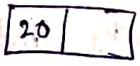
Keys: 20, 40, 30, 15, 45, 5, 60

given order: 3

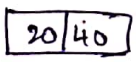
max ptrs = 3 min ptrs = $\lceil 3/2 \rceil = 2$

max nodes = 2 min nodes = 2-1 = 1

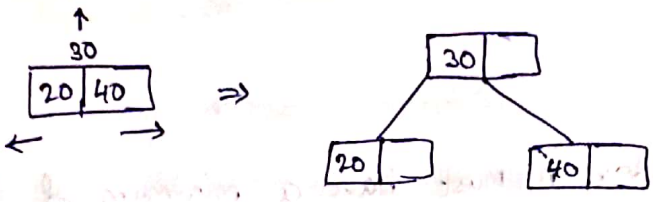
20:



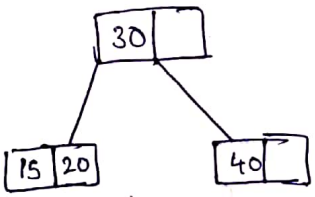
40:



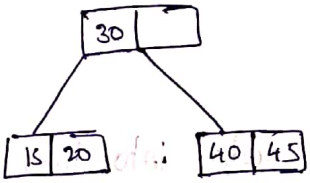
30:



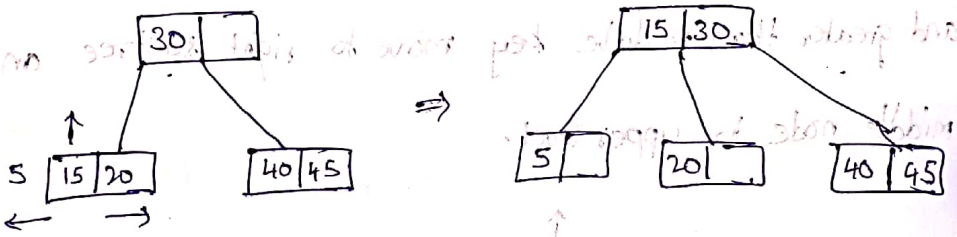
15:



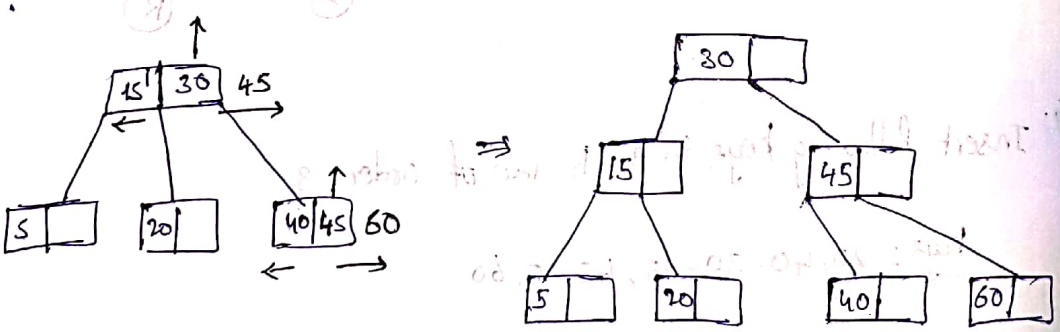
45:



5:

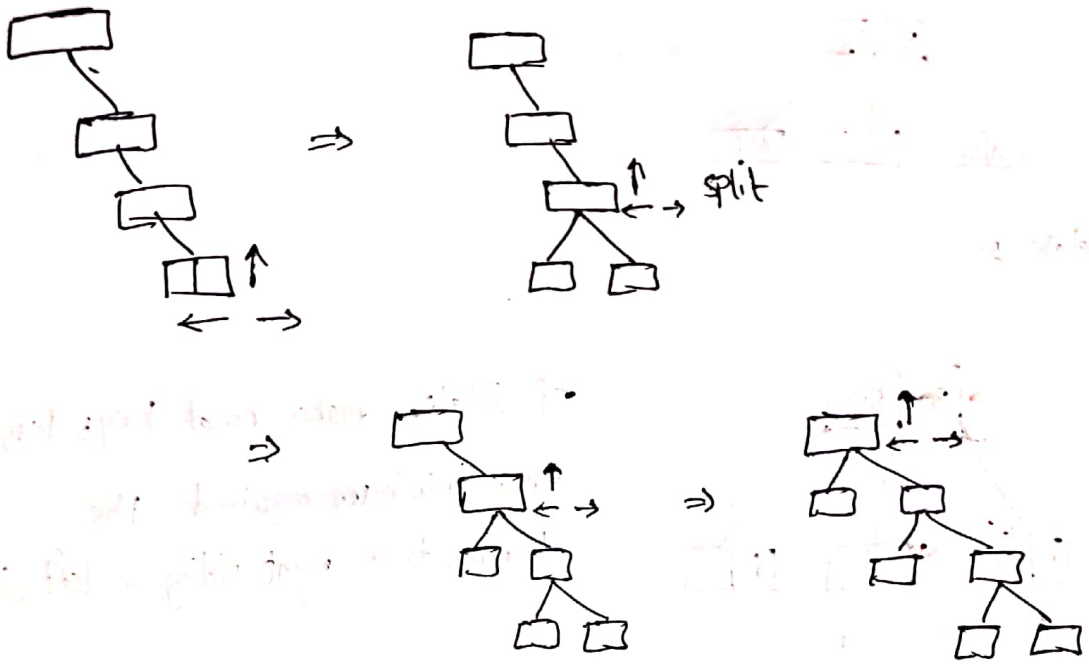


60:



Q 6-05 A B-Tree used as an index for a large database table has four levels including the root node. If a new node is to be inserted, then maximum no of nodes that could be newly created in the process are _____

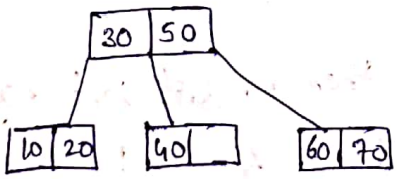
Sol:



∴ no of new nodes = 5

Deletion:

Consider

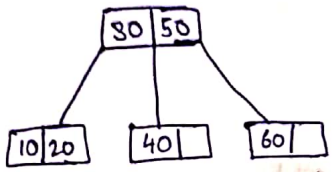


order = 3

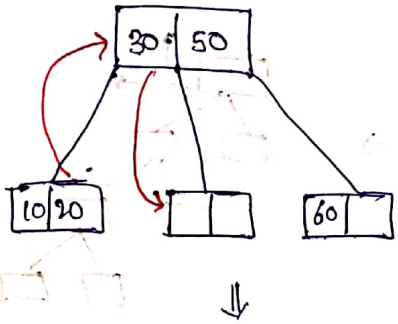
max: 3P, 2k

min: 2P, 1k

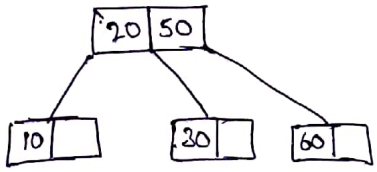
Delete 70:



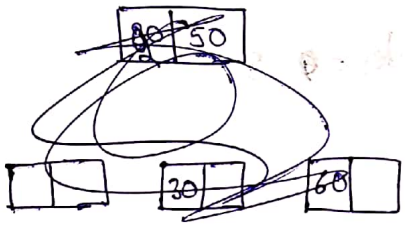
Delete 40:



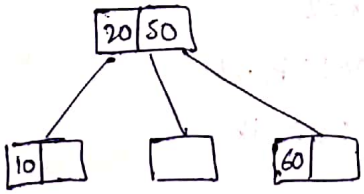
If deletion makes no of keys less than minimum required, the borrow from right sibling or left sibling



Delete 10:

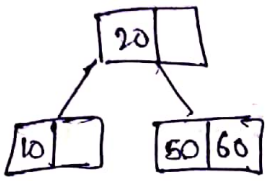


Delete 30:

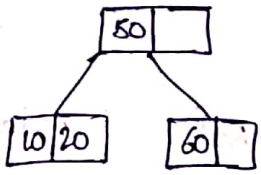


Here there no sufficient keys to borrow from left or right sibling. Thus we borrow from parent and merge the node from left sibling or right sibling

Borrowing 50

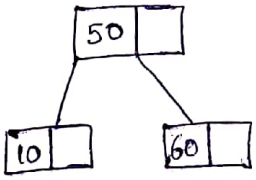


Borrowing 20



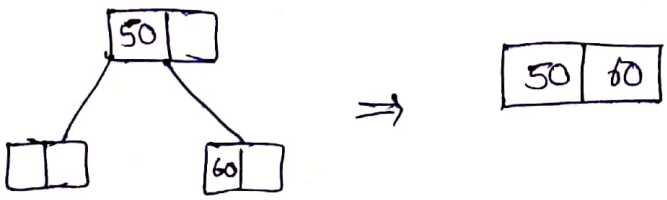
Delete 20 :

when an internal node is to be deleted, replace it either with largest key of LST or smallest key of RST. Here LST node has only one key, so we replace 20 with smallest node of RST.



Delete 10 :

when 10 is to be deleted, we need to check if borrowing from left sibling & right sibling is possible. But it is not possible here. Also borrowing from parent is not possible. so we merge the nodes

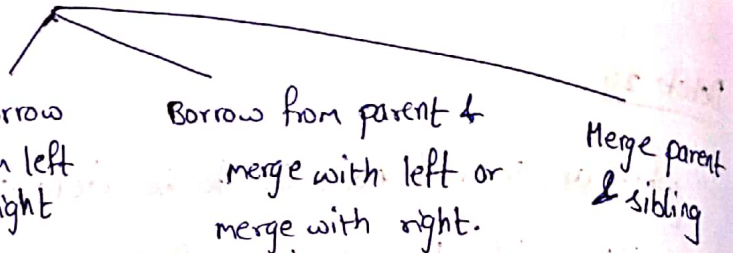


Note:

→ Deletion of leaf

(i) sufficient nodes → no modification

(ii) Insufficient



~~Deletion~~

→ Deletion of internal node:

Replace key value with largest of LST or smallest of RST

Note:

→ Leaf nodes of B-Tree doesn't have any children but still we allocate space for tree pointers in leaf node. This is wastage of space.

→ This drawback is overcome by B+ Tree.

B^+-Tree:

→ It is a modification of B-Tree

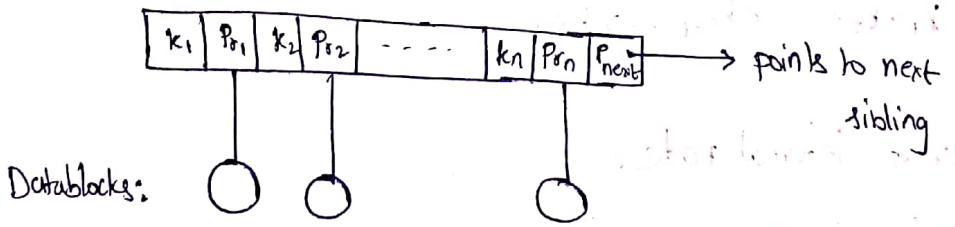
* leaf nodes don't have tree pointers
* Internal nodes don't have record pointers.

leaf node: (key + record ptr) internal node: (key + tree ptr)

→ B^+-Tree doesn't provide direct access.
It provides ordered access.

→ This modification reduces height and increases width (i.e., no of keys in each node)

Structure of leaf node:

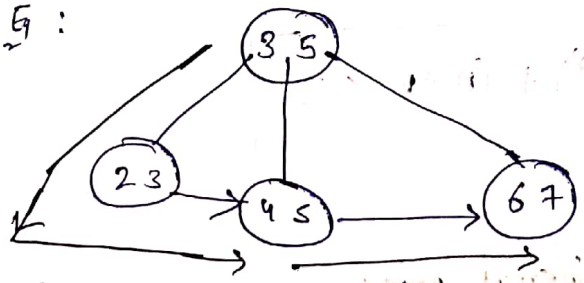


k_1 to k_n ← keys ; $Pr_1, Pr_2 \cdots Pr_n$ → record / data ptrs

The pointer P_{next} is used for range queries

P_{next} → block ptr (or) index ptr (or) tree ptr

Eg:



Query: obtain nodes where $k > 2$ & $k \leq 6$

no of block access = 4

without P_{next} it would be 6

order of leaf node:

If order is n,

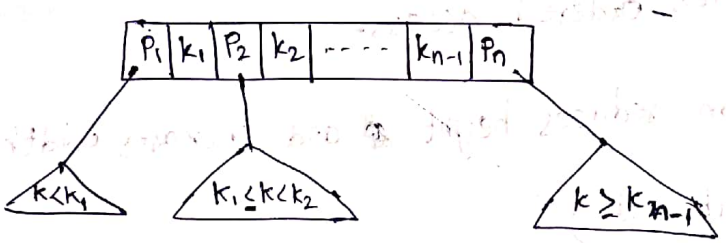
then max no of index records possible = n

If B is size of block, then

$$n*(k+P_r) + P_{next} \leq B$$

Order of leaf node may be different from order of internal node

Structure of internal node:



when key is equal, it has to move to left or right. But same convention has to be followed for entire tree

$P_1, P_2, \ldots P_n \longrightarrow$ Tree/block/index ptrs

$k_1 < k_2 < \ldots < k_{n-1} \longrightarrow$ key

order of internal node:

If order is n,

then max no of tree ptrs = n

If B is size of block then

$$(n*P) + (n-1)k \leq B$$

(P/11) finding order of internal node:

$$n*(6) + (n-1)(9) \leq 512$$

$$15n \leq 521$$

$$\Rightarrow n = 34$$

no of ptrs in 0th lvl → 34

" " " " 1st " → $(34)^2$

" " " " 2nd " → $(34)^3$

⇒ no of nodes in 3rd level = $(34)^3$ [leaf nodes]

order of leaf node:

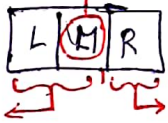
$n*(7+9)+6 \leq 512$

$16n \leq 506 \Rightarrow n = 31$

⇒ no of nodes = $(34)^3 \times 31$

Insertion:

Leaf split:



Actual record of M is kept in the lower level.

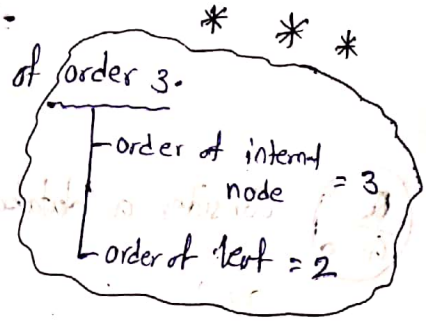
key less than or equal to middle are moved to left subtree. (or) alternate way can also be followed.

Internal split:

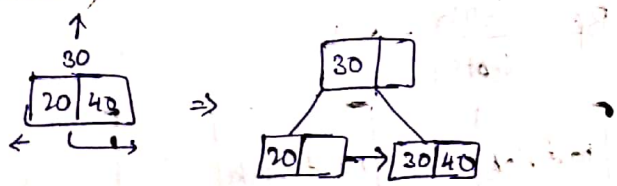
This is same as B-Tree split.

Eg: Insert the following keys in a B^+-Tree of order 3.

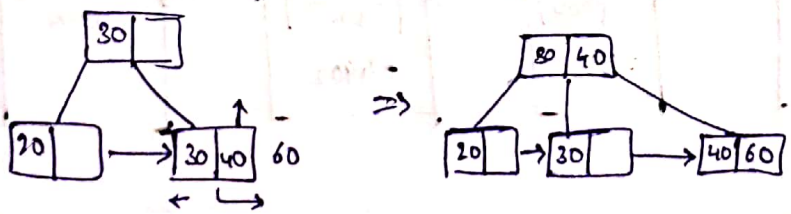
keys: 20, 40, 30, 60, 10, 80, 90



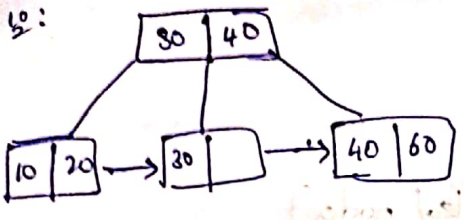
20, 40, 30:



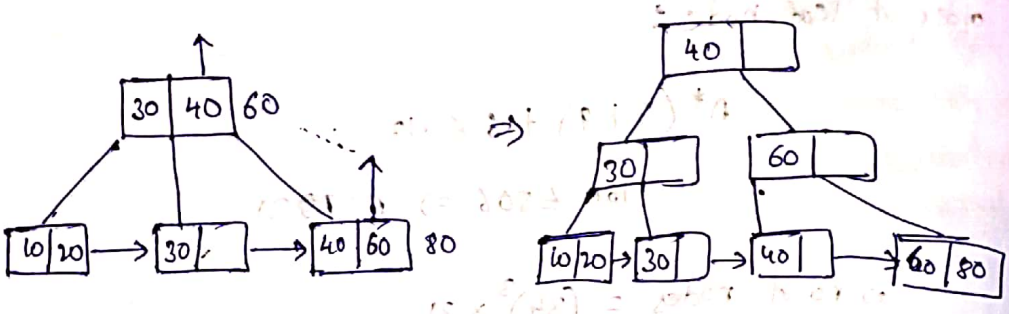
60:



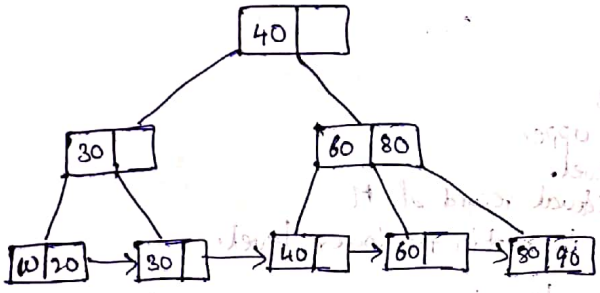
50:



80:



90:



Note:

If order of internal node is n,

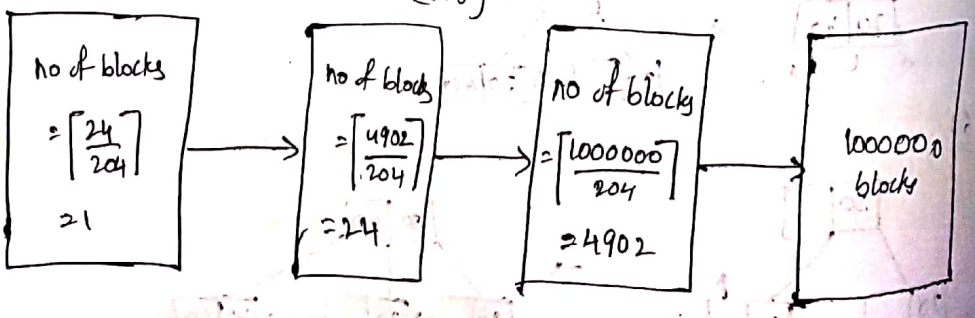
Then min no of ptrs = $\lceil \frac{n}{2} \rceil$

$\Rightarrow$ min no of keys = $\lceil \frac{n}{2} \rceil - 1$

Q G-20 Consider a database ....

Sol:

$Bf = \frac{4096}{(20+8)} = 204$



Method 2:

order of internal node = 205

" " leaf node = 204

lvl	block	ptrs
1	1	205
2	205	$(205)^2$
3	$(205)^2$	$(205)^2 * 204$

no of blocks in leaf level = $(205)^2$

⇒ no of (key + record ptr) in leaf level

$= (205)^2 (204) \approx 10^6$

∴ 4 block accesses

Deletion:

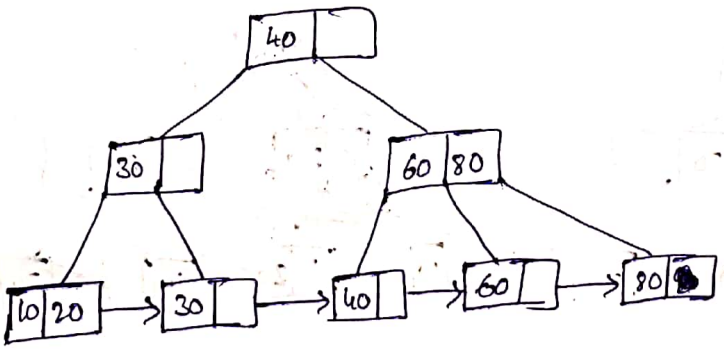
while deletion below points must be checked

(i) height balanced

(ii) minimum order

(iii) Properly (i.e., < .. left node , >= .. right node)

Consider below B+-Tree

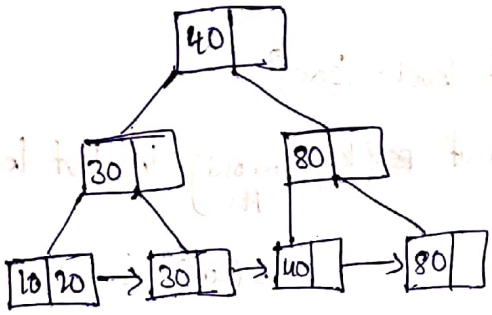


Delete 60:

60 has to be deleted from both leaf & internal node.

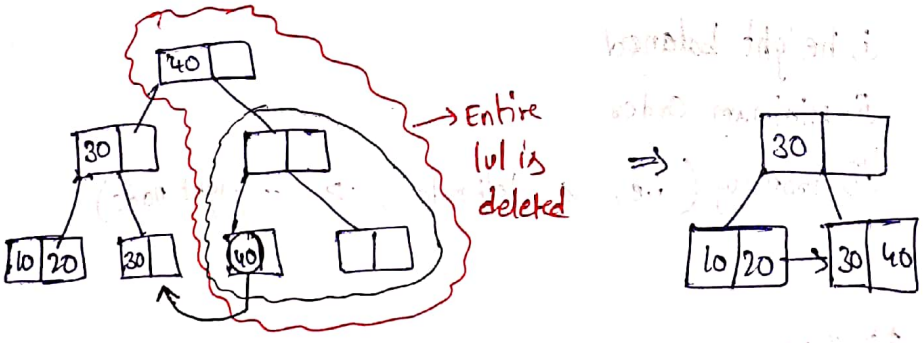
Deleting 60 from internal node, leaves the node empty.

So we replace 60 with 80 (we should not replace it with 40)

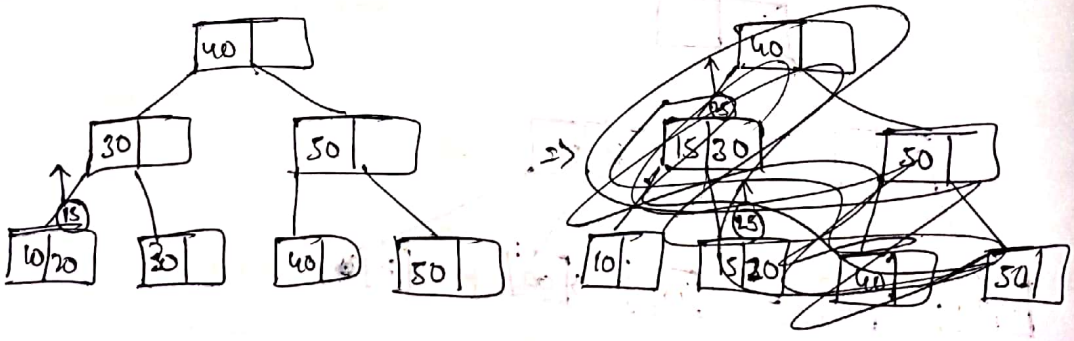


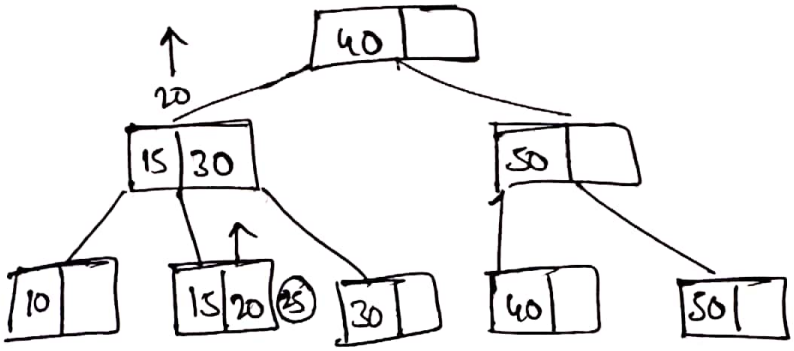
Delete 80:

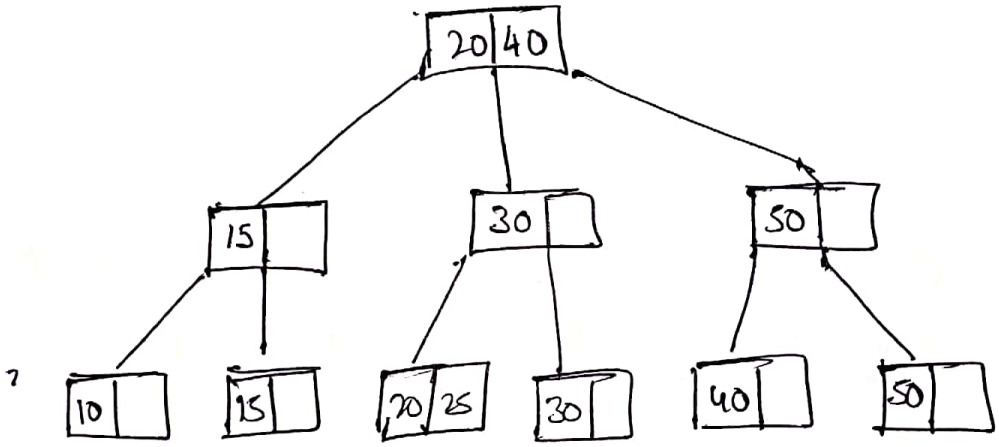
Deleting 80 causes violation of minimum order in both leaf node and internal node. Also there is no possibility of ~~merging~~ borrowing



P/15 15:



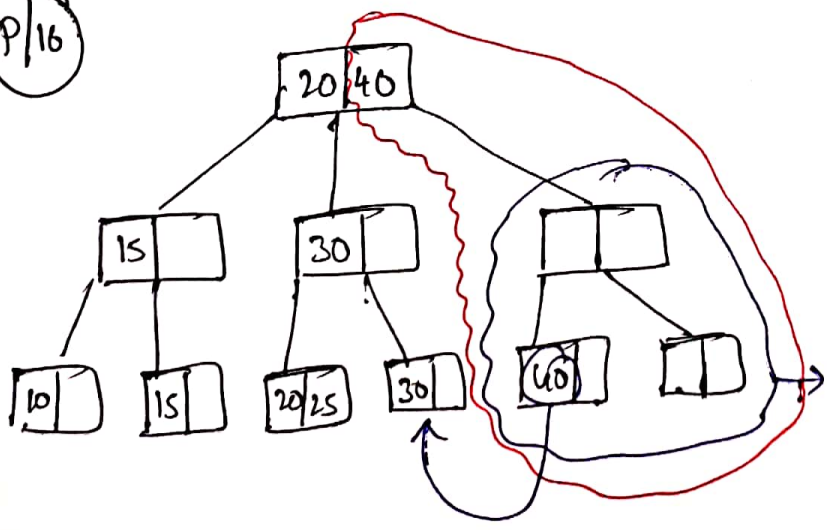




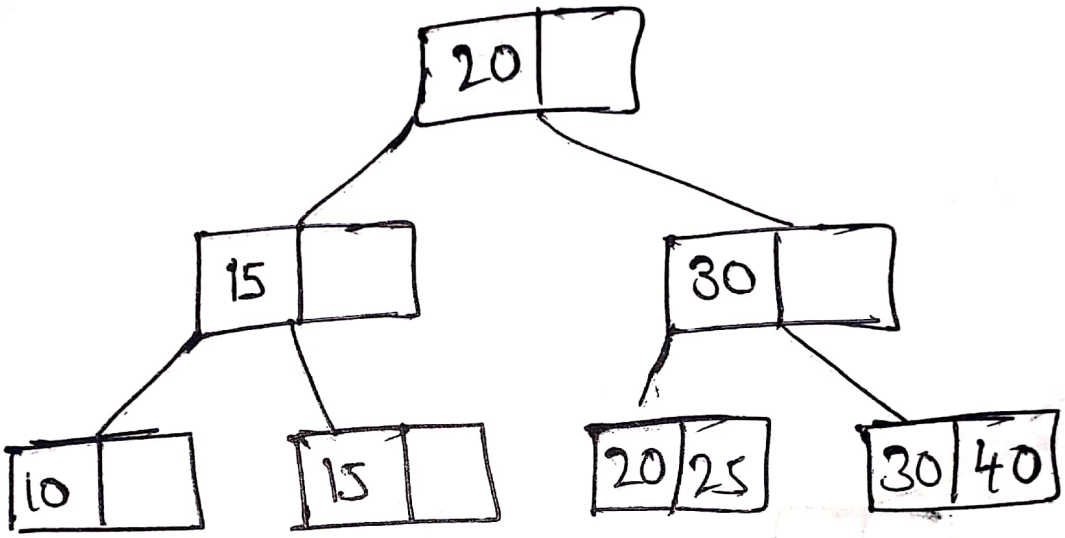
∴ 1

∴ opt-(a)

P/16



⇒ This entire part has to be removed



∴ opt ⓐ