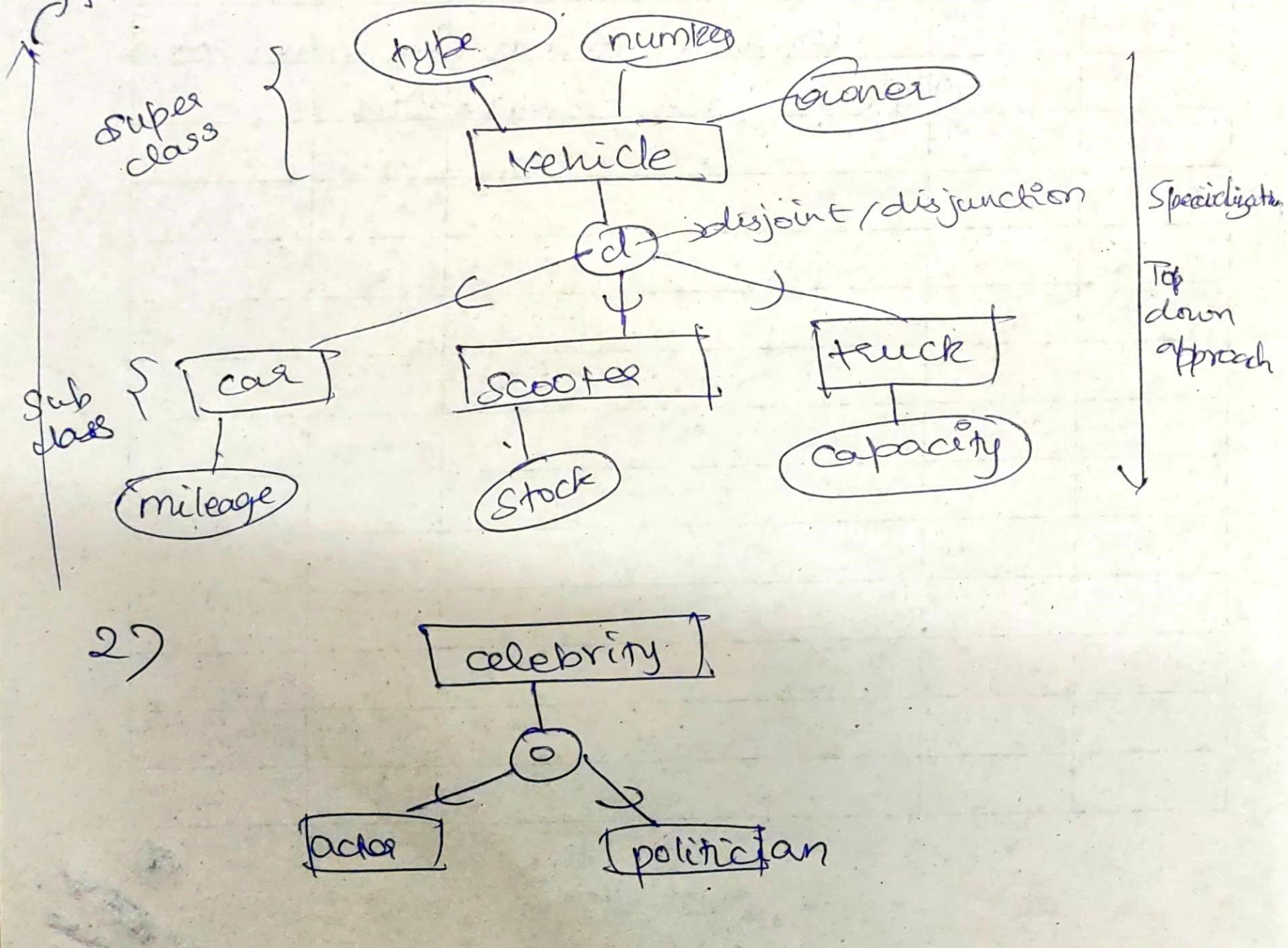
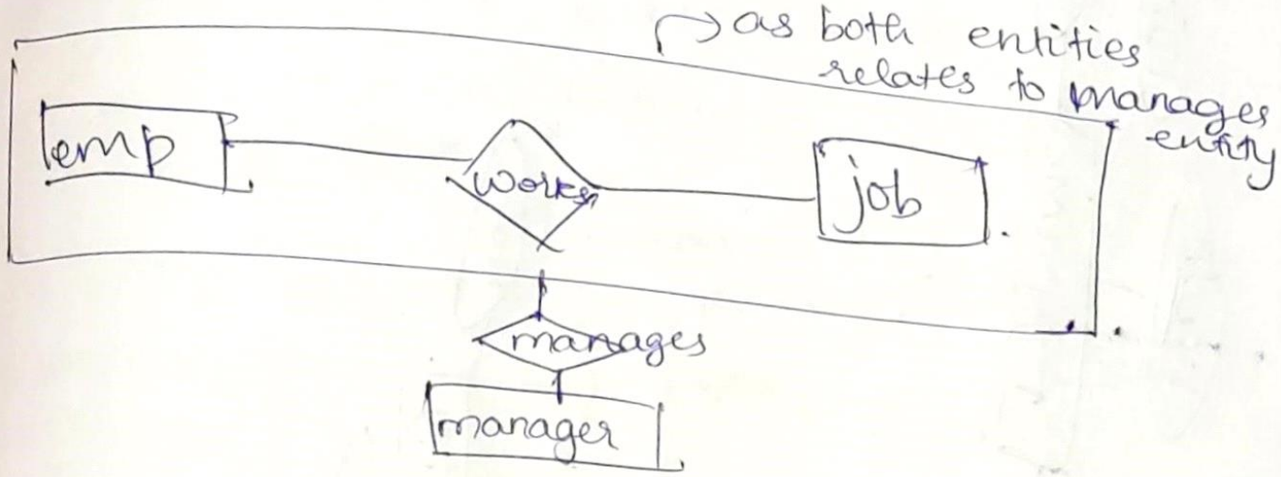
1) EER - enhanced ER diagram



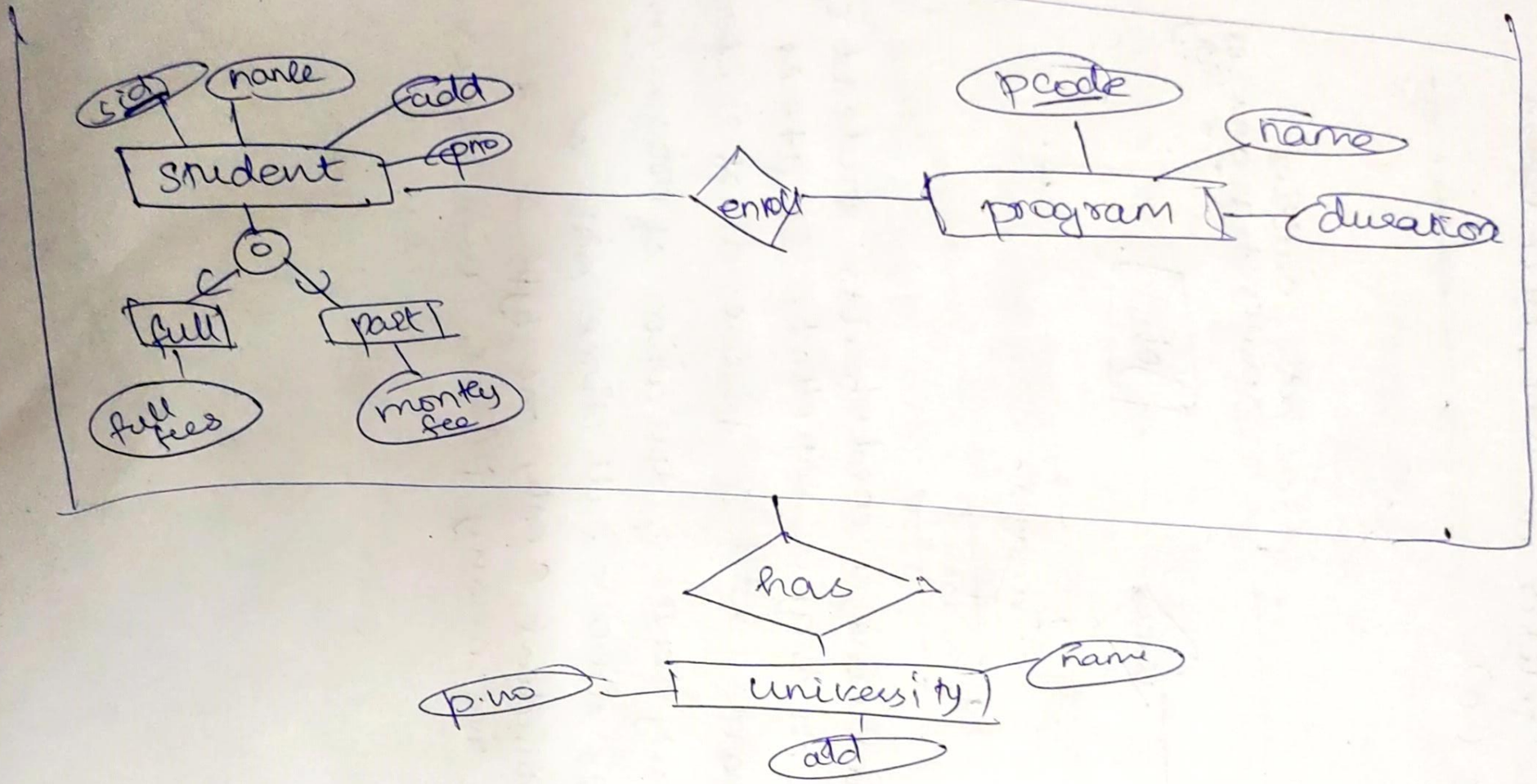
2)

In case 1 the subclasses, if the entities cant be similar (eg: car cant be scooter & vice versa use use disjoint/ disjunction (d). If they can be same eg: actor can be politician or vice versa we use whole (o).

Aggregation:



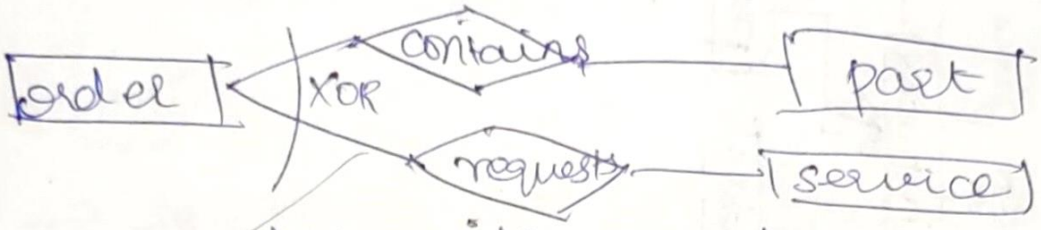
1) A uni maintains record of its student & progs in which they're enrolled. It stores student id, name, add & phone no of student & prog code, prog name & duration of program. A student is either a full time stu or part time student (only one type). A student can reg many progs & vice versa


sid
name
add
student
pno
enroll
pcode
name
program
duration
full
part
full fees
monthly fee

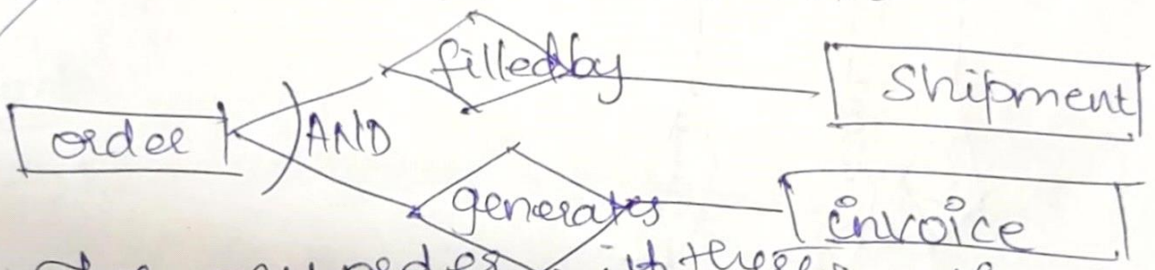
has
p.no
university
name
add

→ EER is conceptual data model.
→ comparable to UML class diagram.
→ also called peter chen's proposal

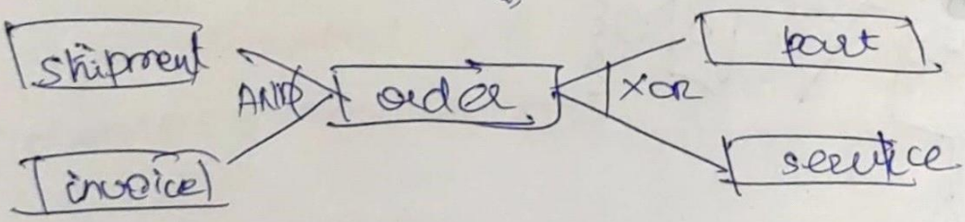
AND / XOR relations

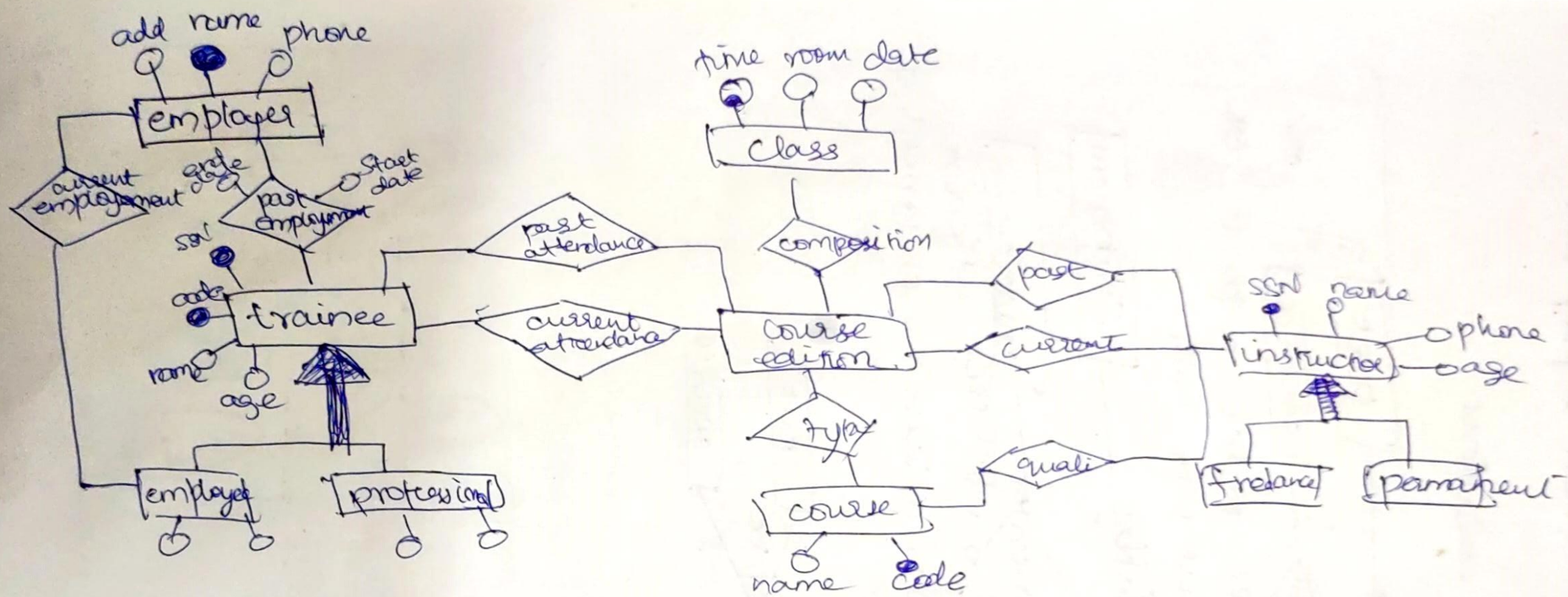


⇒ orders either order a part or request a service, but not both



→ for any order, if there's atleast one invoice there is also @least one shipment & vice versa.




add
name
phone
employer
current employment
grade
start date
past employment
SSN
code
name
age
trainee
past attendance
current attendance
employee
professional
time
room
date
class
composition
course edition
past
current
type
course
quali
name
code
SSN
name
phone
age
instructor
freelance
permanent

Normalization:

↳ decomposing schemas to reduce redundancy

Denormalization:

↳ merging schemas to ↑ redundancy

Partitioning:

→ splitting into multiple tables to reduce I/O cost.

Horizontal:

→ has original columns & subset of original rows.

→ Seperates operational data from archieval data

Vertical:

→ vice versa.

→ used to seperates frequently used columns from infreq used

TUNING:

→ ORDER BY, GROUP BY, joins

SQL Query ⇒ relational algebra:

```
Select Fname, Lname
from Employee where salary > (
select max(salary) from employee
where DNO = 5);
```

divided into 2 parts

$\pi_{Fname, Lname}(\sigma_{salary > \delta_{max\_salary}(\sigma_{DNO=5}(EMP))}(EMPLOYEE))$.

Algo for external Sorting:

Ordered by ⇒ needs to be sorted
→ removes duplicates

Sorting : 2 phases

Sorting ⇒ sort portions of file
merging ⇒ merge sorted portions

* used for faster execution of operation

Internal Sorting: (in MM)

⇒ Quick sort → Bubble sort

External Sorting:

if relation doesn't fit in MM.

External merge sort

1) M blocks (which fits into MM) are read @ a time
2) sorted in memory
3) M blocks are written back

Merge M-1 runs at a time (M-1) way merge:

1) reads first block of M-1 runs.
2) then in O/P 1's record to buffer block
3) continue till buffer block is full
4) O/P buffer ~~block~~ block to disk
5) when block of run is exhausted next block of run is read.

Sorting:

M=3 → buffer can hold 3 records

18, 11, 16, 13, 12, 17, 21, 15, 19, 20, 14

1st run: applying algorithm:

(11, 16, 18) (12, 13, 17) (15, 19, 21) (14, 20)

11, 12, 13

16, 17, 18

14, 15, 19

20, 21

→ and run

Disk: 11, 12, 13, 16, 17, 18 | 14, 15, 19, 20, 21

11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21

Cost of external merge sort:

no. of passes = $1 + \lceil \log_{B-1} [N/B] \rceil$

cost = 2N * (No. of passes)

eg:

with 5 buffer pages, to sort 108 page files

Pass 0 = $\lceil \frac{108}{5 \text{ (B)}} \rceil$ = 21.6 ≃ 22 sorted runs

5 pages in buffer, last run only 3 pags

1 2 3 ... 21 22

5 5 5 5 3

if 5 * 21 = 105 + 3

↳ sorted runs ↳ extra

Pass 1 = $\lceil \frac{22}{(B-1)=4} \rceil$ = 5.5 ≃ 6 sorted runs

20 pages each, last run is 8 pages

1 2 3 4 5 6

20 20 20 20 20 8

⇒ pass 2 : $\left\lceil \frac{6}{3} \right\rceil = 2$ sorted runs

⇒ 80 + 28 = 108

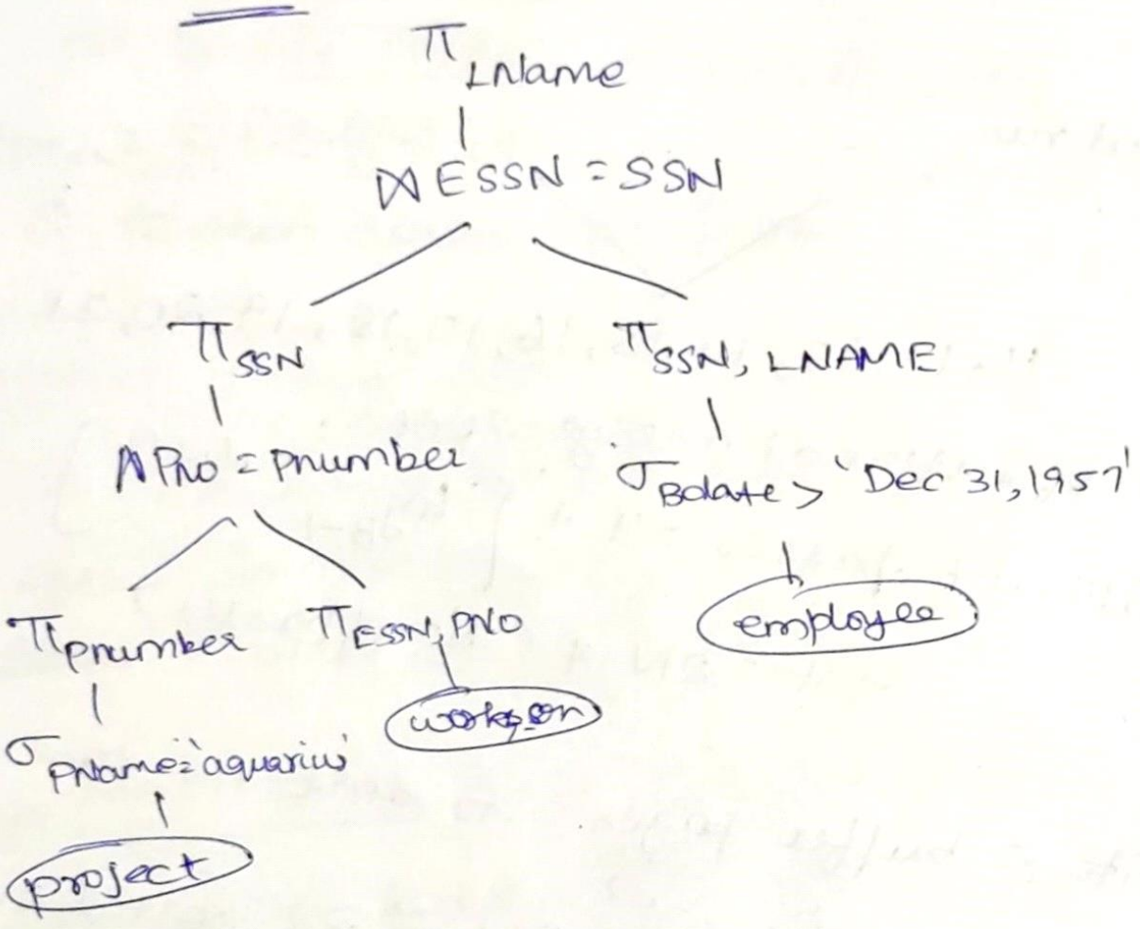
⇒ pass 3 : $\left\lceil \frac{2}{2} \right\rceil = 1$ sorted run

⇓

with 108 pages

So for 108 pages, 3 passes required

Heuristic:



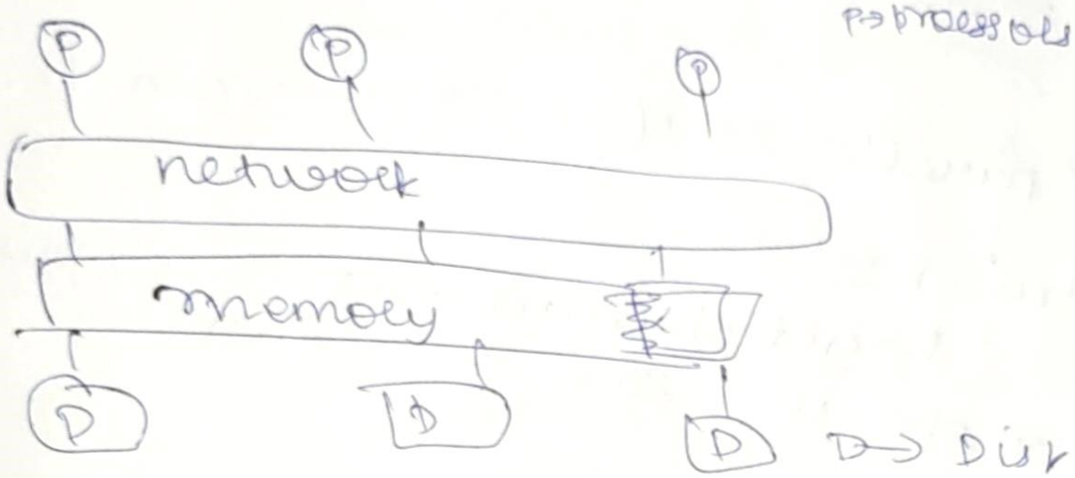
Parallelism

→ uses multi CPU & disk to ↑ throughput & various ops like loading data, evaluating etc. ..

Architectures

→ shared memory
→ shared disk
→ shared nothing
→ heirarchical

Shared memory:



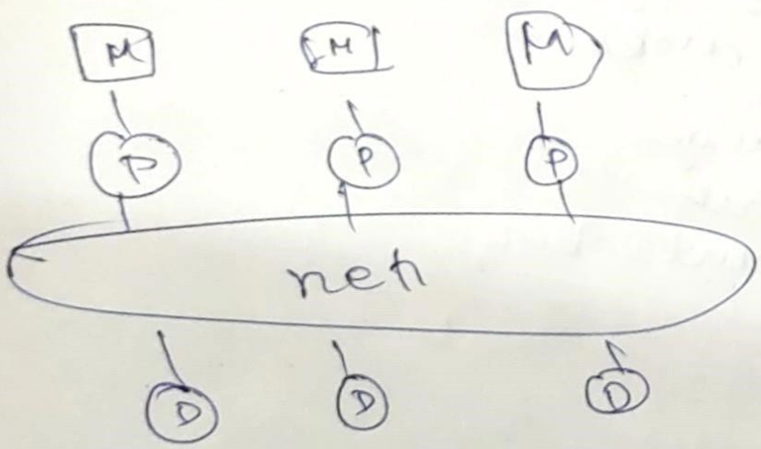
P → Processors

D → Disk

⇒ cache memory ↑, ~~so~~

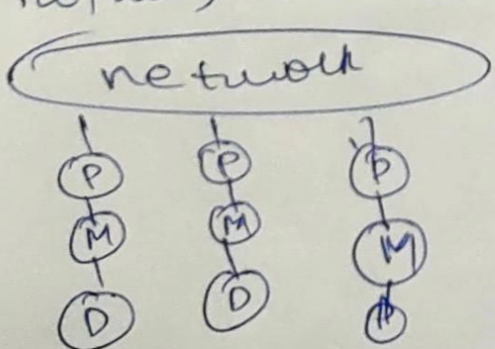
→ ↓ sensitive to partitioning

Shared disk:



⇒ fault tolerance

Shared nothing:



⇒ hard to program

Inter Query: ↑ response time

→ ~~parallel ex~~ no. of transactions (multiple queries run on diff sites) processed in tlle with each other.

Intra Query: ↑ throughput

→ no. of subtasks of transactions

" " " "

(parallel ex of single query run on diff s

Intra operator:

↳ get all machines to compute given operator (Scan, sort, join)

Inter operator:

Data partitioning: [table d_1 d_2 d_3 ...]

1) Horizontal:

→ tuples of relation divided among many disks.

① Range
② Hash
③ RoundRobin.

2) Vertical:

Q

Round Robin : (Spread load)

→ approx same no. of tubles

id	Name	Branch
1		
2		
3		
4		
5		
6		

i = recno = 6

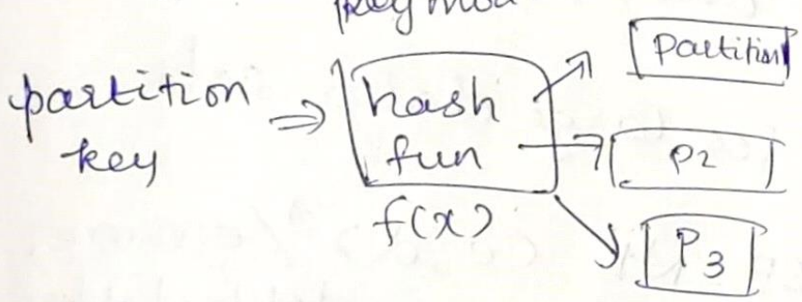
n = no. of disk = 3 (0, 1, 2)

i mod n ⇒ 1 mod 3 → disk 1
2 mod 3 → disk 2
3 mod 3 → disk 0

* only suitable for full table scans
* Specifications ×

Hash : (equijoin, exact match queries)

→



Partition fn can be id, empid etc;

Range : (equijoin, exact match, range queries)

based on att values

salary :

eg: 100 - 5000 in d1

5001 - 10000 in d2

Partition key - salary

< 10000 ⇒ P1

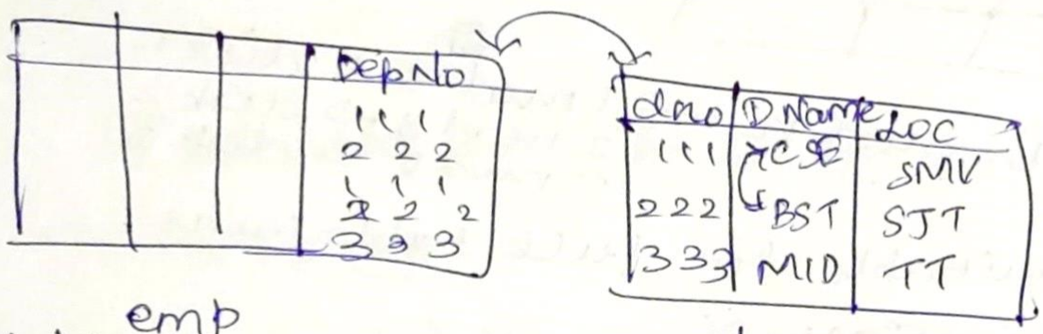
10000 < 30000 ⇒ P2

> 30000 P3

Joins in DBMS

employee ⋈ dep (dno = depno)

→ nested loop joins
→ hash join
→ sort - merge join



emp dep

Nested: In nested loop, ~~use~~ ~~fo~~ for every val of dep no check dno, if yes match

→ 2 for loops
→ not suitable for large data sets

Select /*+ USE_NL (e, d) */ e.name, d.name from employee e, dept d where e.no = d.no;

Hash: (equi join) (=)

build phase: takes table with lesser data & applies hash key on the join key & computes hash table

probe phase: takes other table & matches with hash table

Build:

hash val	Dno	Dnm	D_loc
1	111	ESE	SJT
0	222	BST	SMV

} 2 entries

Join key ⇒ hash fun (key mod n) ⇒ hash value

eg: 111 % 2 = 1 ⇒ so hash val 1
222 % 2 = 0 ⇒ " " " 0

Probe:

Take emp table:

same 111 % 2 = 1 ⇒ so match this hash val with that of build & match.

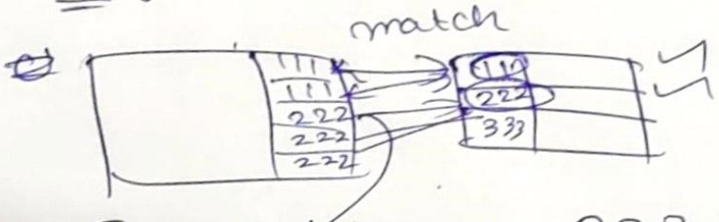
Sort Merge Join

→ Non equi join (< / >) (>= or <=)

1) Sort:

Sort acc to join key (order) (111 → 222 → 333)

2) merge:



In third case, 222 is not matching with key 111, so we increment val of key, then matches.