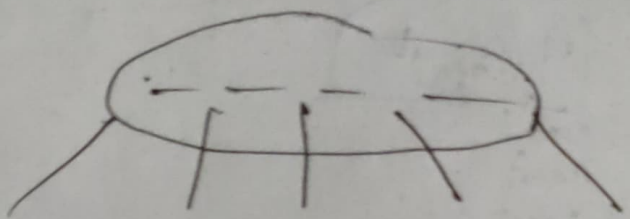
22/03

Binary search tree

order = 1

(may not be balanced)

Multiple search tree



N+1 child

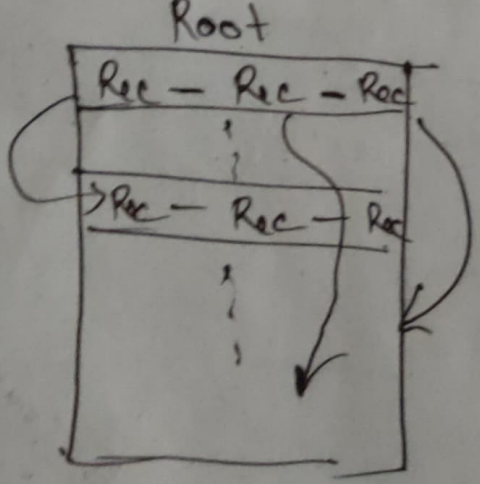
B Tree / B+ tree ⇒ balanced multiway search tree

How B+ tree helpful

→ K ~~nodes~~ keys in 1 node ⇔ 1 block

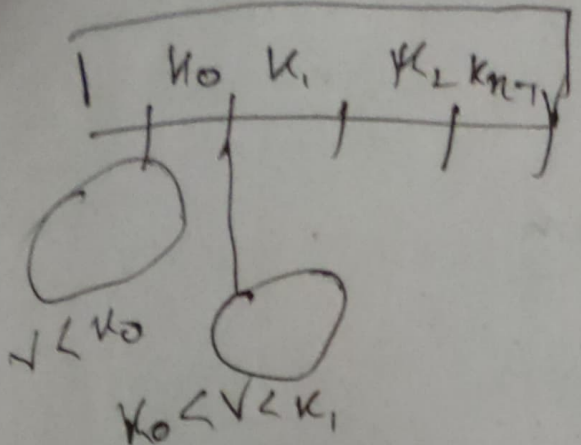
→ K keys are sorted → no of comparisons are same

1 key in 1 block is wastage



In the file

B tree > multiway since B, B+ tree is balanced.



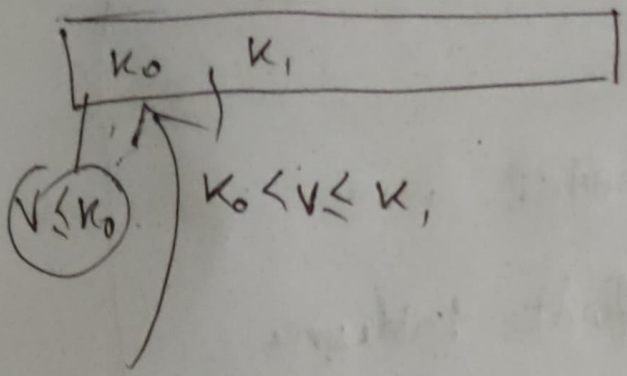
← if n no of keys in B tree

⇓

| n data rec | block ptr + (n+1) subtree pointers |

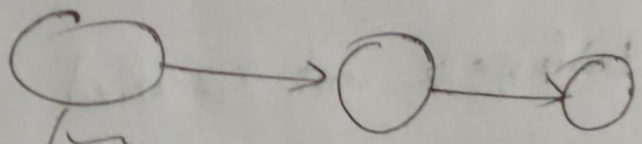
for B^+ tree

↳ equality present



only values are present, no need to keep record pointers

All data records appear at the leaf level



connected as well

| n keys + n rec ptrs + 1 ptr to next leaf |

50% usage is ensured in nodes of B & B^+ tree

⇓

So it is balanced (WHY DF ??)

Depth of B^+ tree can be less

Security

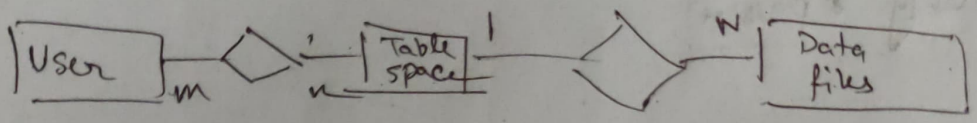
multiuser env

↓

valid user who can work with system

~~DA~~ DBA create user

<CREATE USER userid IDENTIFIED BY pwd
DEFAULT TABLESPACE ——>



CREATE TABLE ⟶ TABLESPACE name

if tablespace not specified → default tablespace

(or) ALTER USER userid
IDENTIFIED BY newpwd
DEFAULT TS newts.

What a system can do??

system level privileges
(Granted by DBA)

obj level priv
(table, view, index)
<Granted by owner of the object>

- connect (to DB) (minimum)
- ~~can't~~ can't create (unless permitted) ←
- Resource level (can create own object)
- DBA

<Grant priv1, priv2 TO userid1 userid2>

<Remove priv1 FROM USERid2>

obj ← SELECT | INSERT | DEL | REFERENCE

GRANT priv1, priv2, .. ALL
ON objectname TO userid1, id2 | PUBLIC
with GRANT OPTION

REVOKE ___
FROM ___

tablename

Restrict No of bytes

GRANT RESOURCE (10 M) ON TABLESPACE tsname
To userid1, userid2

int M/K

if 0 → remove access.

View obj (logical table)

CREATE VIEW viewname AS (SELECT * FROM ___)

View
- → single table
- → multitable

INSERT into myview
VALUES ___
SELECT * from myview
___

DROP view viewname
(doesn't affect table)

select, insert, delete, update
depends on situation
it may not work.
Eg insert in (select Roll from ___)

SE view

Select *
where D= 'CSE'
WITH CHECK OPT
↑ so that D='EC' is not added

Query processing

27/3

DML statement to DBMS

↳ ~~not~~ non procedural

↓

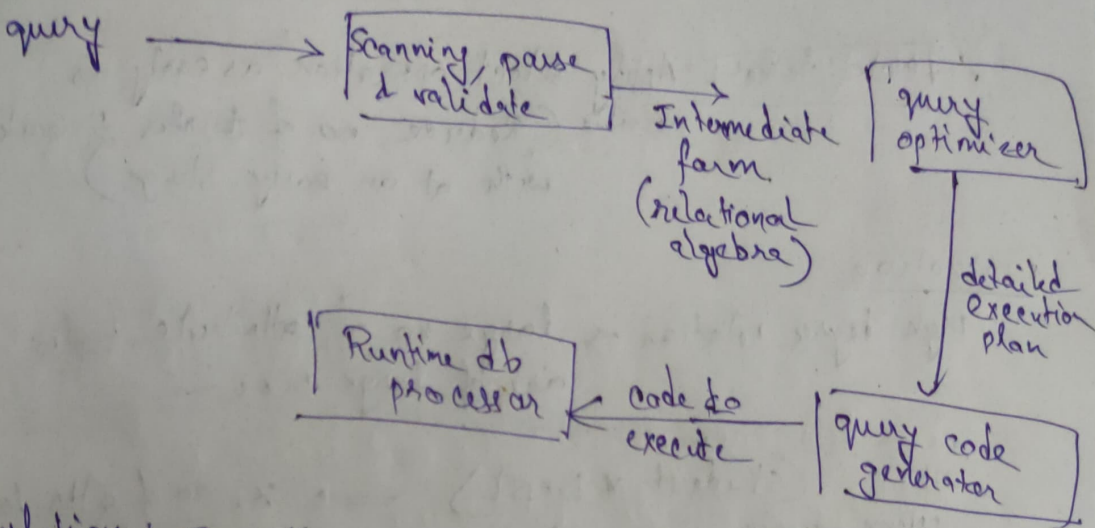
how to execute it actually & efficiently ⇒ ~~Det~~ Detailed procedure

Given a non-procedural query making a detailed & efficient but equivalent representation is the task of query processor. Formalting the efficiency strategy

↓

query optimization

Translation optimization



Translation: Scan the query, parses the same, to obtain the tokens, validates the relation & attribute (checks the syntax) ⇒ used the data dictionary

View name ⇒ replaced by view definition

finally generates equivalent expression in eq form

optimisation: 2 phases

① Expression level optimisation: Expr ⇒ equivalent and efficient (may not be the best) form

② Detailed strategy.

Eg: Select op.

$\sigma_{DNAME='CSE'}$ (Student ⋈ Dept) → how to do ① ??

(Try to reduce disk block access)

N_1 N_2 → deal with $N_1 * N_2$ tuples !!

(Student ⋈ $\sigma_{DNAME='CSE'}$ DEPT) → deal with $N_1 * 1$ tuples !!

$\sigma_{DNAME='CSE'}$ $\wedge$ $MONTH(DOB)$ $(Student \bowtie Dept)$

⇓

$(\sigma_{MONTH(DOB)} Student) \bowtie (\sigma_{CSE} Dept)$

∴ Note: select : Apply select operation as early as possible (Reduce no. of tuples to work with at an early stage)

Projection

High degree relation ⇒ large no of attribute tuples are of large size.

$\Pi_{DName, SNAME}(Student \bowtie Dept)$ ⟹ k_1 no of attrs from S, k_2 no of attrs from D

$(k_1 + k_2)$: internal mem req is large!!

⇓

$\Pi_{name}(Student) \bowtie \Pi_{DName}(Dept)$??? How?? : take common ones

$\Pi_{name, dname}(\Pi_{sname, DCODE}(Student) \bowtie \Pi_{DNAME, DCODE}(DNAME\ Dept))$

(Ans)

∴ take the projection as early as possible

↳ required in final o/p & those needed for intermediate processing.

Q) $\Pi_{roll, name, score}(\sigma_{score>80}(Student \bowtie result))$

$\Pi_{name, score}(\Pi_{roll, name}(Student) \bowtie \sigma_{score>=80}(result))$

R1 * R2 * R3

≡ R1 * (R2 * R3) or (R1 * R2) * R3 ??

Eg: Emp * Dept * Nominee

1000 Less ≥ Emp-tuples

~~80~~ 20 1200

do this first

Not this } huge!!!!!!

② Detailed Strategy

- Query: Involves a simple criteria (Based on single attr)
  - → Linear search
  - → If data is sorted on the search attr.
    - ↳ go for binary search.

Looking for a specific key:
- → Search attr is a key attr & primary index exist use primary index to get the record.

→ Search attribute is non-key clustering index exists use it to get all the desired rec.

→ Secondary index on key → useful to retrieve based on key attr.

Non key-attr ⇒ search attr ⇓ sec. index

Use suitable index file for Range query (>=, <=, ≠)

Complex criteria (involving multiple attr)

Conjuncture (AND)

Disjunctive (OR)

if index exists
↳ union of all the indexes

→ for every simple criteria ⟹ make the strategy
↳ for one condition: index search is possible
⇓
retrieve the record
(less no of recs, on which verify other criteria)

→ For few cases efficient (except linear search) strategy is possible ~~to obtain~~
apply indiv. to obtain set of rec
↳ intersection
↳ go for verific to reject.

TITU

Join: R1 ⋈ R2 ← n2 tuples
n1 tuple

Simple iteration

for each tuple $t_1 \in R1$:
  for each tuple $t_2 \in R_2$:
    check (t_1, t_2) → decide

↳ $n1 * B2$ block accesses
↑ no of block access req to read n2

or swap
↳ $B2 + n2 * B1$

keep smaller one in outer loop

∴ total

$B1 + n1 * B2$ (Best case)

$n1 + n1 * B2$ (worst case) ← (all in diff blocks)

if one is very small

↳ keep all tuple in memory (R2)