Page No.
Date. 07/09/23

SQL Clause

SELECT [DISTINCT] A1, A2 ... An [ ] ← Optional Clause.
FROM R1, R2 ... Rn
[WHERE P]
[GROUP BY ATTRIBUTE [HAVING Condition]]
[ORDER BY Attribute [DESC]]

Query Execution:

1. **FROM Clause**: It is the first executable clause. It just simply relation or Cross Product of two or more relation.
2. **WHERE Clause**: It is the second executable clause. It select the tuple based on specified condition.
3. **GROUP BY Clause**: It is the third executable clause if used in the query. It groups the table based on the specified attribute.

Q1. Why SQL is most widely used?

Ans. Because Relational Algebra works on 'Set Theory' and SQL works on multi-set (Bag) where duplicate are allowed

eg: * Set [1, 2, 3, 4]
* multiset [1, 1, 2, 3, 3, 4]

Q2. Commercial Database works on Multiset (Bag) why?

Ans If we want to perform Union of two table (Relation), SQL just simply One table and simply add the data of another table (∵ it's fastest). But in relation we first eliminate duplicate then perform operation (very time consuming).

* In R.A it eliminate the duplicate so we cannot find the answer of multiple tuple value and removing duplicate is expensive
* In SQL aggregate functions are allowed but it not allowed in relational algebra!
* SQL have lots of feature that are not supported by R.A

Page No.
Date. / /

eg: Supplier (Sid, Sname, Rating)
Parts (Pid, Pname, Color)
Catalog (Sid Pid, Cost)

WAQ to find sid of Red Color Parts:
(Rename)
→ SELECT C.Sid AS Krishna FROM Catalog AS C, Parts P WHERE C.Pid = P.pid AND P.Color = Red.

WAQ to find Sname of Red Color Parts:
→ SELECT S.Sname FROM Supplier AS S, Part P, Catalog AS C WHERE P.Pid = C.Pid AND C.Sid = S.Sid AND P.color = Red.

Note: Whenever we want multiple copies of the same table to be used then we need to use "Rename/Aliasing".

Aggregate Function: The function that takes collection of value as a input (from set or multiset) and produce single output. It always discards the Null values first.

1. COUNT
2. SUM
3. MIN
4. MAX
5. AVG

eg.

Sid	Branch	Marks
S1	CS	90
S2	IT	70
S3	CS	70
S4	EC	56
S5	CS	NULL

Group by Branch →

Sid	Branch	Marks
S1	CS	90
S3	CS	70
S5	CS	NULL
S2	IT	70
S4	EC	56

High frequency tuple will be at top

COUNT(marks) = 4
COUNT(*) = 5 (# of tuples)
COUNT([DISTINCT]marks) = 3

SUM(marks) = 286 (Null is removed)
SUM([DISTINCT]marks) = 216
AVG(marks) = 71.5

eg:

A
NULL
NULL
NULL
NULL
NULL.

COUNT(*) : 6
COUNT(A) : 0
SUM(A) : NULL
AVG(A) : NULL
MIN(A) : NULL
MAX(A) : NULL

Important Points About Aggregate Functions:

* Arithmetic Operations with NULL gives result "NULL".
  , NULL * Any value = NULL
  (Arithmetic Operator) (O/P)

* NULL : Not existed / Unknown
  - Two values are not equal
  - Non zero
  - Random ASCII values assigned by DBMS S/W.

* Aggregate Operator always discard/ignore the NULL values. While Performing the aggregate operations then first discard or ignore the NULL values then performed computations

* Comparisions Operations with NULL gives Result "Unknown".

HAVING: * Fourth executable Clause (if used in query), after "Group By"
* It is used to select the group which satisfy the Conditions (Condition is for each group)

Sid	Branch	Marks
S1	CS	60
S2	IT	70
S3	CS	90
S4	IT	60
S5	EC	55
S6	EC	NULL

Group By Branch →

Sid	Branch	Marks
S1	CS	60
S3	CS	90
S2	IT	70
S4	IT	60
S5	EC	55
S6	EC	NULL

```
SELECT * FROM Student
GROUP BY (Branch)
HAVING AVG(MARKS) > 65
```

Sid	Branch	Marks
S1	CS	60
S3	CS	90
S2	IT	70
S4	IT	60

Page No.
Date. / /

Note: If aggregate operator and other attribute (normal or unaggregate) used in Select Clause is allowed only iff other attribute (non-aggregate attribute) must be in Group By Clause.

SQL SET Operators

1. UNION & UNION ALL
2. INTERSECT & INTERSECT ALL
3. MINUS & MINUS ALL

[EXCEPT] followed by Relational algebra

[EXCEPT ALL] Not followed by R.A.

* If "HAVING" is applied without "GROUP BY" Clause then HAVING is applied to every tuple according to latest standard of SQL.

eg:

R

1
1
1
2
2
4
6
6

S

1
1
2
4
5
5

R U S =

1
2
4
6
5

R UNION ALL S =

1
1
1
2
2
4
6
6
1
1
2
4
5
5

R INTERSECT S

1
2
4

R INTERSECT ALL S

1
1
2
4

R MINUS S

6

R MINUS ALL S =

1
2
6
6

Other SET Operators:

1. IN / NOT IN
2. ANY
3. ALL
4. EXISTS / NOT EXISTS

Comparision Operators

<, >, <=, <>

↳ not equal

Page No.
Date. 08/09/23

ANY Operator : Compare a value with each value in a set and return true if any value is compared according to given condition.

Nested Queries

1. Normal (Independent) Nested Query.

Execution Sequence
Bottom → Top
Inner Query → Outer Query

2. Correlated Nested Query.

Execution Sequence:
Top → Bottom → Top
Outer Query → Inner Query → Outer Query

eg: Retrieve Sid and marks of the student who secured Higher marks?

Query I: Select Sid max(marks) From Student.

Query II: Select Sid, max(marks) From Student Group By Sid;

Query III: Select Sid marks
From Student
Where marks = [Select max(Marks) From Student]
(Outer Query) (Inner Query)

Note: Aggregate functions cannot be in lowercase.

ANY Operator : Compare a value with each value in a set and return true if any value is compared according to given condition

eg: ANY (OR)

x > ANY (10,20,40) O/P ⇒ 11, 12, 13, 14 .....

(x>10) or (x>20) or (x>40)

x < ANY (10,20,40) O/P ⇒ 39, 38, 37, 36 .....

Page No.
Date. / /

ALL Operator: Compare a value with each value in a set and return true if given condition satisfied for each value in the set.

eg: $x > ALL(10, 20, 30)$ O/P: 31, 32, 33, ...

$x > 10$ AND $x > 20$ AND $x > 30$

Q1. Find the name of Supplier whose turnover is better than the turnover of any (some) Supplier of Delhi.

SUPPLIER

Sno	Sname	City	Turnover
1	A	Delhi	4 Cr.
2	B	Bangalore	5 Cr.
3	C	Delhi	6 Cr.
4	D	Kochi	7 Cr.

SELECT Sname From Supplier WHERE City <> Delhi AND turnover > ANY (Select turnover From Supplier WHERE City = Delhi)

O/p: ANY [B, D] ALL [D]

Q2. Retrieve eid who get more salary than any employee of dept = 5?

EMP

Eid	dno	Salary
E_1	5	50K
E_2	3	20K
E_3	5	30K
E_4	3	40K
E_5	4	60 K.

Query I: SELECT Eid FROM Emp WHERE dno <> '5' AND Salary > ANY (SELECT Salary From Emp WHERE dno = '5')

Query II: (Without ANY Operator)

SELECT Eid From Emp WHERE dno <> 5 AND Salary > (Select min(Salary) WHERE dno = '5').

O/p: ANY [E_4, E_5] ALL [E_5]

Q3. Consider the following relation Cinema (theatre, address, capacity) Which of the following options will be at the end of the SQL Query "SELECT P1.address FROM Cinema P1" such that it always finds the addresses of theatres with maximum capacity?

Ans WHERE P1.Capacity >= ALL (Select P2.Capacity from Cinema P2).

Page No.
Date: / /

Supplier (Sid, Sname, Rating)
Parts (Pid, Pname, Color)
Catalog (Sid, Pid, Cost)

Supplier

Sid	Sname	Rating
S1	Abhay	8
S2	Mukesh	9
S3	Riya	9
S4	Pooja	9
S5	Alok	9

Catalog

Sid	Pid	Cost
S1	P1	6k
S2	P2	7k
S3	P3	8k
S4	P4	9k

Parts

Pid	Pname	Color
P1	A	Red
P2	B	Green
P3	C	Red
P4	D	Yellow

1. WAQ to find S.ID of Supplier of whose rating > 8
→ SELECT Sid FROM Supplier WHERE rating > 8 Or
SELECT T.Sid FROM Supplier T WHERE T.rating > 8

S2
S3
S4
S5

2. WAQ to find S.id whose Parts Cost greater than 7k.
→ SELECT X.Sid FROM Catalog X where X.Cost > 7000
Or SELECT Sid FROM Catalog WHERE Cost > 7000

S2
S4

3. W.A.Q to find Pid of Red color parts
→ SELECT Pid FROM Parts WHERE Color = 'Red'

P1
P3

4. W.A.Q to find Sid of Supplier who Supplied Red Color Parts.
→ SELECT C.Sid FROM Catalog C, Parts P WHERE C.Pid = P.pid
AND P.color = 'Red'.

S1
S2

5. W.A.Q to find 'SName' who Supplied Red color parts.
→ SELECT S.Sname
FROM Catalog C, Parts P, Supplier As S
WHERE C.Pid = P.pid AND
C.Sid = S.Sid AND
P.color = 'Red'.

Sname
Abhay
Mukesh

Page No.
Date. / /

6. W.A.Q to Find Sid who not supplied Red Color Parts?

→ SELECT C.SID FROM Catalog C, Parts P WHERE C.pid = P.pid AND P.colour <> 'Red'.

Sid
S1
S4

Note:

X = 5 R[1, 2, 3, 4, 5]

X IN R

'5' IN [1,2,3,4,5] ⇒ True

X NOT IN R

'5' NOT IN [1,2,3,4,5]

⇒ False

Query: SELECT Sid FROM Catalog WHERE Pid IN (SELECT Pid FROM Parts WHERE Color = Red).

Using "IN"

(i) P1 IN (P1, P3) : True
↳ then S1 Selected

(ii) P2 IN (P1 P3) : False
↳ Select nothing

(iii) P3 IN (P1, P3) : True
↳ P3 S2 selected

(iv) P4 IN (P1, P3) : False
↳ nothing Selected.

Color = Red

Ans

Sid
S1
S2

Using "NOT IN"

P1 NOT IN (P1 P3) False

P2 NOT IN (P1, P3) : True
↳ S1 Selected

P3 NOT IN (P1, P3) : False

P4 NOT IN (P1 P3) : True
↳ S4 Selected

Sid
S1
S4

⇐ Color nor Red

Homework: (Using the Supplier, Parts, Catalog table)

1. WAQ to find Sid who Supplied Some Red Color parts or Some Green Color parts.
2. WAQ to find Sid who Supplied Some Red Color parts and Some Green Color parts

STUDENT (Id, Name, Branch, Marks)

3. WAQ to Retrieve Id of Student who secured higher marks (For highest marks) in → R.A, SQL, max (Aggregate) function.
4. WAQ to Retrieve Id of Student who secured 2nd higher marks.
→ (i) R.A (ii) SQL (iii) Max (Aggregate) function

Page No.
Date. / /

Supplier (Sid, Sname Rating)
Parts (Pid Pname Color)
Catalog (Sid Pid Cost)
} Relations

(i) W.A.Q to find Sids who supplied some red color parts or some Green color parts

RA: $\Pi_{Sid}\left[\begin{array}{l}\sigma_{C.pid=P.pid \wedge (Catalog \times Parts)}\\ P.Color = Red \vee Green\\ P.color = Red \overset{or}{=} \vee P.color = Green\end{array}\right]$

SQL: SELECT Sid
FROM Catalog C, Parts P WHERE C.pid = P.pid AND P.color = 'Red' OR 'Green'.

(ii) W.A.Q to find Sids who supplied some 'Red' Color parts and some 'Green' color parts.

RA: $\Pi_{Sid}\left[\begin{array}{l}\sigma_{C.pid=P.pid \wedge}\\ P.color='Red'\end{array}(Catalog \times Parts)\right] \cap \Pi_{Sid}\left[\begin{array}{l}\sigma_{P.pid=C.pid \wedge}\\ P.Color = Green\end{array}(Catalog \times Parts)\right]$

SQL:

```
Select C.Sid From Catalog C, Parts P WHERE
C.pid = P.pid AND P.color = 'Red'
INTERSECT
Select C.Sid FROM Catalog C, Parts P WHERE
C.pid = P.pid AND P.color = 'Green'.
```

STUDENT

Sid	Name	Branch	Marks
S1	Kavebi	CS	80
S2	Dwarsh	ME	72
S3	Renchir	CS	92
S4	Bhavya	IT	90

(i) W.A.Q to find retrieve sid of student who secured higher (1st higher marks). marks

O/P:

Sid
S3

RA: $\Pi_{Sid}(Student) - \Pi_{sid}\left[Student \underset{marks<m}{\bowtie} \rho_{I,N,B,M}(Student)\right]$

Page No.
Date. / /

SQL: SELECT Sid FROM STUDENT
EXCEPT
SELECT T1.Sid FROM Student as T1, Student T2 WHERE
T1.marks < T2.marks

With MAX function: SELECT Sid FROM Student WHERE
Marks = [SELECT max(marks) FROM (Student)]

(ii) W.A.Q to retrieve Sid of Student who secured 2nd highest marks: O/P = S4

RA $\rho\left(Temp, \Pi_{Sid\ marks}\left[Student \bowtie_{marks<M} \rho_{I,N,B,M}(Student)\right]\right)$

$\Pi_{Sid}(Temp) - \Pi_{Sid}\left[Temp \bowtie_{marks<m} \rho_{I,m}(Temp)\right]$

SQL:

Used to Sub query o/p as a db table

```
WITH Temp (Sid, marks)
    SELECT T1.Sid T1.marks FROM Student T1, Student T2
    WHERE T1.marks < T2.marks
SELECT Sid FROM Temp
EXCEPT
SELECT T1.sid FROM Temp T1, Temp T2
WHERE T1.marks < T2.marks
```

Using Max Function:

```
SELECT Sid
FROM Student WHERE MARKS = [ SELECT max(marks)
                             FROM Student
                             WHERE marks < [ Select max(marks)
                                             From (Student) ] ]
```

Page No.
Date. / /

Q1. The following relation records the age of 500 employees of a company where empNo {indicating the employee number} is the key.
empAge (empNo, Age)
Consider the following R.A expression:
Π_{empNo} (empAge $\bowtie$ (age>age1) $\rho_{empNo, age1}$ (empAge))
What does the above expression generate?

Ans Employee numbers of all employees whose age is not the minimum.

Q2. Retrieve Sid and Marks of the Student who secured highest marks

× Query I: Select Sid max(marks) } Syntax error
From Student

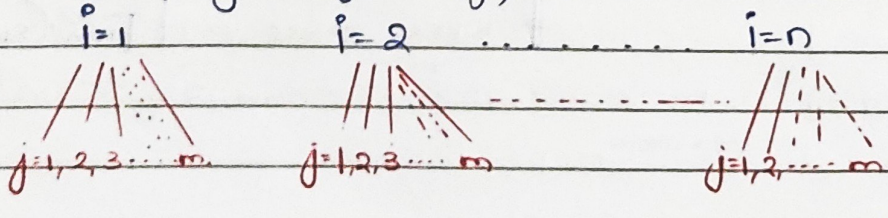
× Query II: Select Sid, max(marks) } Syntax correct but it gives all Sid and their marks, not giving topper id and marks.
From Student
Group By (Sid)

✓ Query III: Select Sid marks
From Student
Where marks = (Select max(marks) From Student)

Note: * When aggregate operator and other attribute (non aggregate) used in the Select Clause is allowed only if other attribute must be in the group by clause.
* Aggregate functions cannot be in lower clause.

Co-related Nested Query Execution: Inner Query using attribute defined in outer query.

```
for(i=1, i<=n; ++i)
    for(j=1; j<=m; ++j)
```



Page No.
Date. / /

eg: of Co-related nested query:

```
SELECT C.sid
FROM Catalog C
WHERE EXISTS ( SELECT * FROM Parts P
               WHERE P.pid = C.pid
               AND Color = 'Red' )
```

Inner Query using attribute defined in outer query

EXISTS (Checks): Return True if Inner Query Result Not Empty

NOT EXISTS : Return True if Inner Result Empty

Execution of above Query:

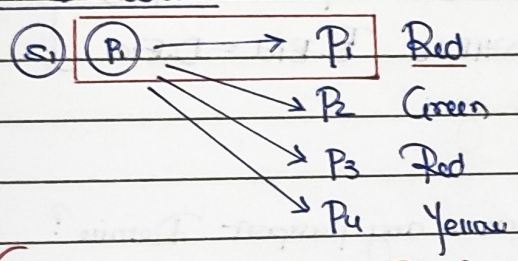
Catalog

Sid	Pid
S_1	P_1
S_2	P_2
S_3	P_3
S_4	P_4

Parts

Pid	Color
P_1	Red
P_2	Green
P_3	Red
P_4	Yellow.

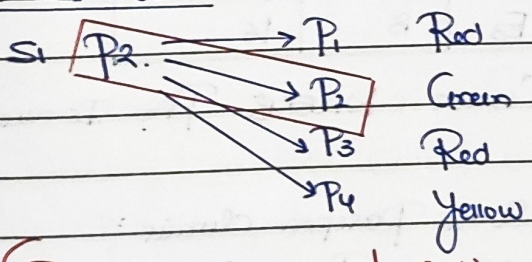
1st Iteration



(Pid match & color = Red)

⇒ 1 Tuple Return.

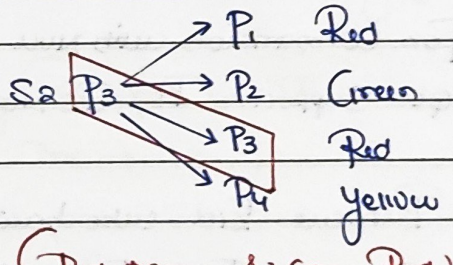
2nd Iteration



(Pid match but color != Red)

⇒ 0 Tuple return

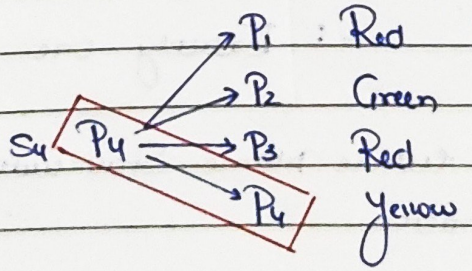
3rd Iteration



(Pid match & Color Red)

⇒ 1 Tuple Return

4th Iteration



(Pid match but color didn't match)

⇒ 0 Tuple Return

EXIST: Color = 'Red' → S1, S2

NOT EXIST: Color != 'Red' → S4

* Before EXIST and NOT EXISTS no attribute is required.
* Before IN & NOT IN Attribute is required.

Q3 Given relational Schema Emp(Eid, Ename, Salary)
Department(Eid, dname, Code)
Retrieve Employee Id who have no Department.

× Query I: Select Eid From Emp E, Dep D
where E.Eid <> D.eid

✓ Query II: SELECT Eid FROM Emp E WHERE
NOT EXISTS [SELECT * From Dep D
WHERE E.Eid = D.Eid]

Ans Q2 is Correct & Q1 is wrong

Q4 Retrieve Eid who does not have any passport Details?

→ SELECT Eid
From Emp
WHERE Ppno IS NULL

eid	ename	PassPort no
E1	A	20
E2	B	NULL
E3	B	16

* For comparision with NULL SQL Support IS / IS NOT Clause WHERE Ppno IS NULL

(ii) Retrieve Eid who having Some Passport details:
→ SELECT Eid FROM Emp WHERE Ppno IS NOT NULL

Regular Expression: % : Zero or more Character
_ : Exactly one Character.

Q5 Retrieve Student whose Name Start with 'S' & end with M & at least 5 Characters.

Page No.
Date. / /

Ans S ___ % m.

→ SELECT * FROM Student WHERE Sname LIKE 'S ___ %m'

LIKE: It is used to compare to specify certain search condition for a pattern in a column.

Q6. Retrieve all Students whose name not start with 'C'?

Ans SELECT * FROM Student WHERE Sname NOT LIKE 'C%'.

eg: SELECT * FROM Student WHERE Sname
LIKE A% → Starts with A
%J → ends with J
%I% → Contains I
'___' → All 4 length name
'S__' → Start with S and exactly 4 length
'S___%' → Starts with S and atleast length is 4.

1. R × S : Cross Join
2. R ∪ S : Union Join
3. R ⋈ S : Inner Join
4. R ⟗ S : Outer Join

ORDER BY : Order by clause is used to Sort the rows
*By default : Ascending [ASCE]
Descending [DESC]

Q7. Consider the relations : Supply (SupplierId, Itemcode) with 1000 tuples
Inventory (Itemcode, Color) with 2500 tuples
Let p and q be the number of maximum and minimum records in Supply Join inventory, the value of p+q is ___. (Itemcode is FK in Supply table).

Ans Supply : Referencing Relation
F.K : Item code
Maximum No. of Tuples : 1000
Minimum No. of Tuples : 1000

Inventory : Referenced Relation
Here Itemcode is key so Itemcode cannot be NULL
So min. tuple = 1000

Page No.
Date. / /

Join with foreign key Concept.

(i) R(ABC) S(BCD)

B is foreign key

1000 Tuple → 5000 Tuple

Maximum Tuples in RMS = 1000

(ii) R(ABC) S(BCD)

B is foreign key

1000 Tuple → 4000 Tuple

Maximum Tuples in RMS = 1000

Note:
* The value present in foreign key must be present in primary key of referenced relation.
* Foreign key may contain "duplicate" and "null" values
* Table which contains foreign key ⟶ Referencing Relation (Table)

* Whenever Two Table (Relation) are Joined (natural join) with respect to primary key and foreign key then maximum number of tuples in the resulting relation is equals to number of tuple in the referencing relation.

Q8. Consider the relations:
Employee (Eid, Ename) with 2000-tuples
Department (Eid, Did) with 1000 tuples.
Let p and q be the number of maximum and minimum records in Employee Join Department. The value of 'p+q' is 2000. (Eid is Fk in department table).

Ans Maximum = 1000
Minimum = 1000
} 2000 Total

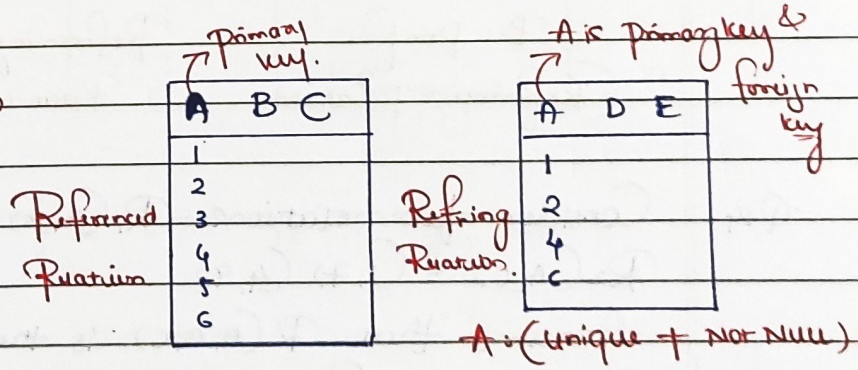
Page No.
Date. / /

Q9. Consider the relations:
$R_1(A, B, C)$ with m tuples.
$R_2(B, D, E)$ with n tuples (B is FK in R_1)
Maximum tuples in $R_1 \bowtie R_2$ = n
Minimum tuples in $R_1 \bowtie R_2$ = n (if B is foreign key & not null)
Minimum tuples = 0 (B is not FK, B is not key, or B is having Null values)

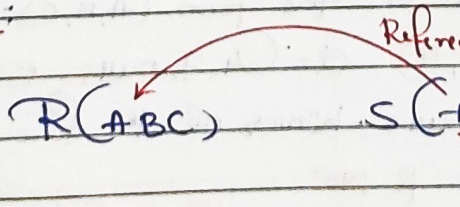
Q10. Consider the relations:
$R_1(A, B, C)$ with m tuples
$R_2(B, D, E)$ with n tuples (B is not FK in R_1)
Maximum tuples in $R_1 \bowtie R_2$ = n
Minimum tuples in $R_1 \bowtie R_2$ = 0

Q11. Consider the join $R \bowtie S$ between $R(ABC)$ and $S(ADE)$ with attribute A being the primary key in both relations and attribute A in S is a foreign key referring to attribute A in R. R has m tuples and S has n tuples, maximum and minimum tuples in $R \bowtie S$ respectively are?

Ans min(m,n), min(m,n)



Note:



Here, the value of A in Relation S contain not only {NOT NULL} but not duplicates also.

A in S: Must be Unique + Not Null But those value of 'A' must be present in 'A' of Relation 'R'. (Foreign key Concept)

Page No.
Date. / /

Q12. Consider the relations:
$R_1(A, B, C)$ with m tuples
$R_2(D, B, E)$ with n tuples $\{A \to B, B \to C, D \to B, B \to E\}$
Maximum and Minimum tuple in $R_1 \bowtie R_2$.

Ans $R_1(ABC)$ $R_2[DBE]$
$[A]^+ = [ABC]$ $[D]^+ = [DBE]$ D: primary key
A is P.K.

→ B is non key attribute in both relation R & S
Maximum = m x n
Minimum = 0

Q13 The following functional dependencies hold for relations R(A,B,C) and S(B,D,E) FD for both the tables. $\{B \to A, A \to C\}$
The relation R contains 200 tuples and the relation S contains 100 tuples. What is the maximum number of tuples possible in natural join R ⋈ S?

Ans
R(A B C) S(B D E)
B: Primary Key (Referred Relation) Referencing Relation
Min no of tuples = 100

Q14 Consider two relations $R_1(A, B)$ with the tuples (1,5), (3,7) and $R_2(A, C) = (1,7), (4,9)$
Assume that R(A,B,C) is the full natural outer join of R_1 and R_2. Consider the following tuples of the form (A,B,C): a = (1,5,null), b = (1, null, 7), c = (3, null, 9), d = (4, 7, null), e = (1,5,7), f = (3, 7, null), g = (4, 9, null). Which of the following statements is correct?

Ans. Relation contains e, f, g but not a, b

R ⋈ S
R ⟗ S

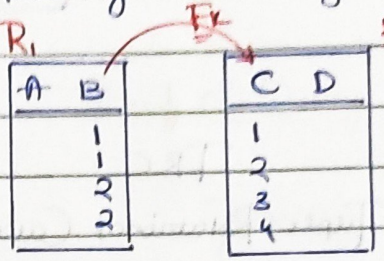
A	B	C	
1	5	7	e
3	7	N	f
4	N	9	g

Page No.
Date. / /

Qis Suppose $R_1(A, B)$ and $R_2(C, D)$ are two relation schemas. Let r_1 and r_2 be the corresponding relation instances. B is a foreign key that refers to C in R_2. If data in r_1 and r_2 satisfy referential integrity constraints, which of the following will always be true?

Ans $\Pi_B(r_1) - \Pi_C(r_2) = \phi$

$[1,2] - [1,2,3,4] = \phi$



Q16. Consider the following database schema named water_Schemas: The no. of tuples returned by the following SQL Query is:

```
WITH total(name, capacity) as
SELECT dist_name, sum(capacity)
from water_schemas
GROUP BY dist_name
WITH total_avg(capacity) as
SELECT avg(capacity)
From total
SELECT NAME
From total, total_avg
WHERE total.capacity > total_avg.capacity.
```

Scheme_no	District_name	Capacity
1	Ajmer	20
1	Bikaner	10
2	Bikaner	10
3	Bikaner	20
1	Churu	10
2	Churu	20
1	Dungargarh	10

Ans

Total

Name	Capacity
Ajmer	20
Bikaner	40
Churu	30
Durgargarh	10

Total_Avg

Avg Capacity : 25

Total Cap > Total Avg Capacity

Name
Bikaner
Churu

Ans = 2 Tuple

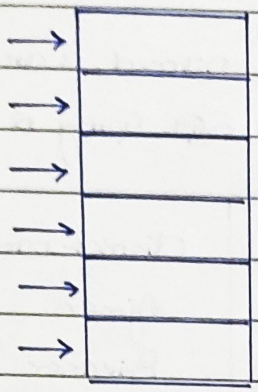
Page No.
Date. 14/09/23

Relational Calculus (Based on Predicate Calculus)

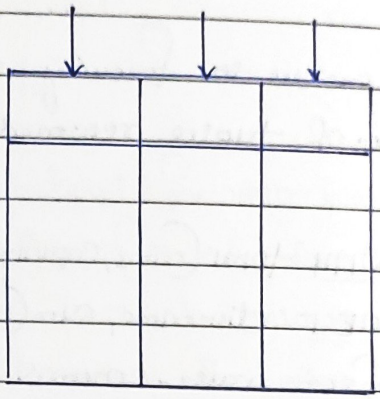
Non Procedural query language and has

2 Variant

- TRC { Tuple Relational Calculus }
- DRC { Domain Relational Calculus }



Tuple as a basic unit.



DRC Takes a Column (Domain) and examine it.

TRC	DRC
Query describe result in the form of set of Tuples.	Query describe result in the form of set of Column (Domain)
Tuple Variable It is a variable that takes on tuples of a relation schema as value.	Domain Variable It is a variable that range over the domain of some attribute (Column).

Form of Query: $\{ T \mid P(T) \}$.

- T ↑ Tuple Variable
- $P(T)$ ↖ Formula which describes Tuple Variable

Page No.
Date. / /

Form of Query: $\{ \langle x_1, x_2, x_3, \ldots, x_n \rangle \mid P\langle x_1, x_2, \ldots, x_n \rangle \}$

- $x_1, x_2, x_3, \ldots, x_n$: Domain variable
- $P\langle x_1, x_2, \ldots, x_n \rangle$: Formula which describe domain variable

TRC

$\{T \mid P(T)\}$

T: Tuple variable

P(T): Formula over Tuple(T) such that P(T) is satisfied.

Form of Query

T{ SELECT Sid

P(T){ FROM Student

WHERE *Condition*.

Note: Formula of TRC is expressed using first order logic. (predicate logic)

First Order Logic:
Variable are of 2 Types

1. Free Variable
2. Bounded Variable

Quantifiers:

1. $\forall$: For all (ALL)
2. $\exists$: There Exists (ANY)

Bounded Variable: If tuple variable is preceded by the quantifier then it is bounded variable.

eg: Belong

1. $\forall \in$ Supplier
2. $\exists \in$ Supplier

Free Variable: If tuple variable is not bounded by the quantifier then it's free variable.

Page No.
Date. / /

Note: The result of TRC should be free variable.

Q1. Loan (Loanno, Branch name, Amount)
Find Loan no of amount above 50000.

Ans TRC:

$$\{T \mid T \in Loan\ (Total\ amount > 50,000)\}.$$

DRC:

$$\{\langle Loanno, branchname, amount\rangle \mid \langle Loan\ no, Branch, amount \in Loan\ (amount > 50000)\}.$$

Q2 Retrieve Sid of the Supplier who supplied some Red color parts.

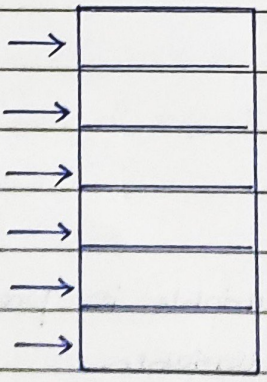
Ans Relational Algebra:

$$\Pi Sid\ (\sigma_{Cpid = P.pid}\ (Catalog \times \sigma_{Color = Red}\ (Parts)))$$

TRC:

$$(T \mid \exists C \in Catalog\ (\exists P \in parts\ (P.color = Red \wedge P.pid = C.pid) \wedge T = C.sid))$$

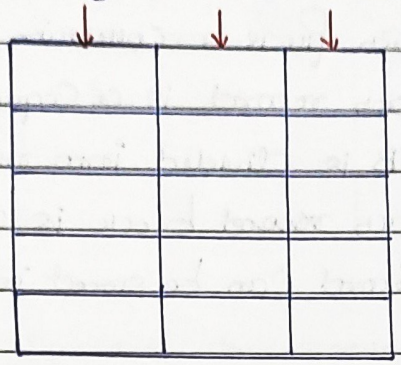
TRC:



→ Takes a Tuple and examine it
→ Tuple as a basic unit

Page No.
Date. / /

DRC: → Takes a Column
→ Range and the column or Domain & examine it.



TRC: Unsafe Operator Occur

Unsafe Operation: Student(t)

~(t): Not belongs to student table (infinite/universe all tuple).
So it is unsafe operation.

Page No.
Date. 11 / 9 / 23

FILE ORGANIZATION & INDEXING

Database
Files
Records
Fields
Blocks

* Database is a collection of files
* Each file is a collection of records
* Each record is a sequence of fields
* DB is divided into number of blocks
* Each record block is divided into records
* Record can be stored in a block

Blocking Factor [BF] : Average number of records per block.

$$\text{Blocking factor} = \frac{\text{Block size}}{\text{Record size}}$$

{ Block Size > Record Size → always greater. }

Organisation

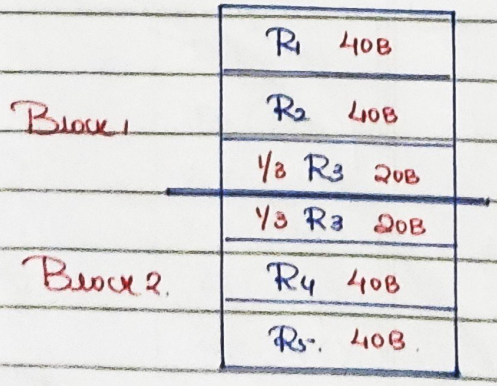
① Spanned Organisation.

→ A record can be stored more than one block (partially can be stored)

eg: Block size = 100B
Record size = 40B

$$\text{Blocking factor} = \frac{100B}{40B}$$

$$= 2.5 \text{ records/block.}$$



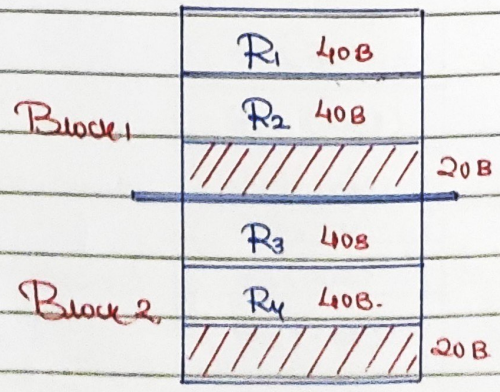
② Unspanned Strategy Organisation.

→ Record can be stored/belongs to a particular block. (cannot stored partially).

eg: Block size = 100B
Record size = 40B

$$\text{Blocking factor} = \left\lfloor \frac{100B}{40B} \right\rfloor = \lfloor 2.5 \rfloor$$

$$= 2 \text{ records/block}$$



Page No.
Date. / /

Spanned Organisation:
- Advantage: no memory wastage
- Disadvantage: block access cost increased
- Suitable for variable length record.

Unspanned Organisation:
- Advantage: Block access reduced, easy manage
- Disadvantage: Wastage of memory (Internal fragmentation).
- Suitable for fixed length record.

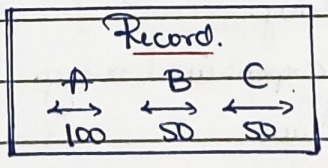
Note:
- In Spanned organisation no memory is wasted but I/o cost is more (block access increase)
- But in Unspanned Organisation memory is wasted but input output cost is less compared to Spanned Organisation.
- Defaut Organisation is Unspanned Organisation.
- **I/O Cost:** Input output cost means number of blocks transfered from a Secondary memory to main memory in order to access some records
- **Search Key:** Attribute used to access the data from DB organisation of records in a file:
  1. Ordered file organisation.
  2. Unordered file organisation.

File Organisation:

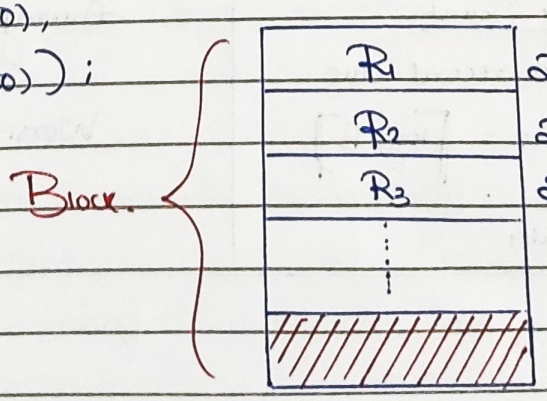
Data Records can be: 1. Fixed length records
2. Variable length records

1. **Fixed length records:**

```
Create table R
(A char (100),
 B char (50),
 C char (50));
```



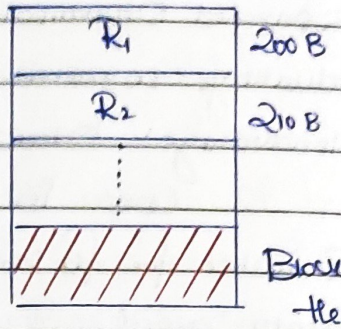
⇒ 200 bytes for each record



② Variable length Records

Create table S
(D char(100),
E char(50),
F text);

∴ Size(D) + Size(E) + Size(text)
100 + 50 + not fixed



DB file with all records fixed length } Both are possible in
DB file with variable length record } RDBMS

Block Factor: Maximum possible records per block.

Block size : B bytes
Block header size = H bytes
Record size : R bytes

→ Block factor for unspanned : $\left\lfloor \frac{B-H}{R} \right\rfloor$ record/block. because for fixed length record unspanned is preferred.

→ Block factor for spanned : $\frac{B-H}{R}$ record/block.

	Ordered File	Unordered File
1.	Searching easy.	Insertion easy
2.	Insertion expensive (re-organising the file)	Searching expensive. Linear Search Average case = $\frac{B}{2}$
3.	Binary Search To access a record avg no. of blocks = $\lceil \log_2 B \rceil$ B: block data	Worst Case = B.

Page No.
Date. / /

File of ordered records (Sorted file)

Ordered (Sequential) file:

* Records sorted by ordering field
* Called ordering key if ordering field is a key field.

Advantages:

* Reading records in order of ordering key value is extremly efficient.
* Finding next record
* Binary search technique. (to access a record avg no of block access $= \log_2 B$)

Files of Unordered Records (Heap Files)

Heap (or pile) file:

* Records placed in file in order of insertion.
* Inserting a new record is very efficient
* Searching for a record required linear search (avg. no of block access $= \frac{B}{2}$).
* Deletion techniques:
  1. Rewrite the Block
  2. Use deletion marker.

eg: Suppose that record size $R = 150$ bytes, block size $B = 512$ bytes, $r = 30000$ records.

Then we get blocking factor: $\left\lfloor \frac{\text{Blocking size}}{\text{record size}} \right\rfloor = \left\lfloor \frac{512\ B}{150\ B} \right\rfloor$

$= 3$ Records / Block

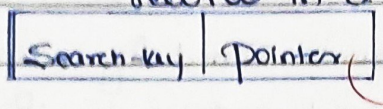
No of file blocks $b = \frac{30,000}{3} = 10,000$ data block.

Indexing:

* Index are used to improve the search efficiency.
* To reduce I/O cost.
* Provide a faster way to access the data.
* Index is an ordered file.

Page No.
Date. / /

* Search key: attribute to set of attributes used to look up records in a file.



→ Points to a block where key is available

* An index file consist of records (called index entries) of the form shown above.

One Index Record Size	=	Size of Search Key	+	Size of Pointer

* Index files are typically much smaller than the original file.
* Two basic kinds of indices:
  1. Ordered indices: Search keys are stored in sorted order.
  2. Hash indices: Search keys are distributed uniformly across buckets using a hash function.
* Index file block size is same as DB file block size.
* Block size of index file = Block size of DB file.
* One index record size = Search Key Size + Size of block pointer.

Note: To access a record average number of block access $= \log_2 B + 1$.

$\log_2 B$ → Index block access

$+1$ → data block access

[Bi: Index Block]

Q1. Given the following data file

EMPLOYEE (Name, Ssn, Address, Job, Sal...)

Suppose that:

* record size $R = 150$ bytes, block size $B = 512$ bytes, $r = 3000$ records
* For an index on the Ssn field, assume that the field size $V_{SSN} = 9$ bytes, assume the record pointer size $P_R = 7$ bytes

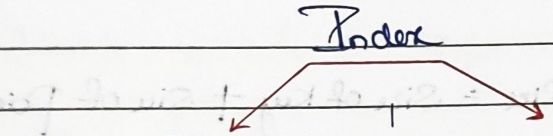
Then

Page No.
Date. / /

Ans. One index record Size = 9+7 = 16 Byte.
Block factor of index file = $\left\lfloor \frac{512\,B}{16\,B} \right\rfloor$ = 32 Index record Per block

Total no. of index entries = 30,000, then total no of index blocks = $\left\lceil \frac{30,000}{32} \right\rceil$ = 938 Index block

Average no. of block access = $\log_2 B_i + 1$
(To access a record) = $\lceil \log_2 938 \rceil + 1 = 10 + 1$
= 11 Block access



* Index entry created for every Search key value
* Number of index entries = Number of DB records

* Index entry is Created for some Search key value.
* Number of index entries = Number of blocks
* Applicable when records are Sequentially ordered on Search key.

In Sparse Index file:
To locate a record with Search-key value k we:
* Find index record with largest Search-key value < k
* Search file Sequentially starting at the record to which the index record points.

Q2 No. of records of file = 16384 (2^{14}). Block Size = 4096 Bytes, Record Size = 256 Bytes. Search key Size = 20 Bytes. Pointer : 10 Byte

(i) If Cost to access record without index based on ____
(a) Ordered file = ____ (b) Unordered file = ____

(ii) Index clause [using] : (a) # of index blocks at 1st level = _____
(b) I/O cost to access record: [using 1st level] = _____

(iii) By using Sparse index:
(a) # index blocks at 1st level: _____
(b) I/O cost to access record: [using 1st level] = _____

Ans: Block factor of DB file = $\left\lfloor \frac{\text{Block Size}}{\text{record Size}} \right\rfloor = \left\lfloor \frac{4096\ B}{256\ B} \right\rfloor = 2^4$ Record per block

Total # data blocks = $\left\lceil \frac{\#\text{records}}{\text{BF of DB file}} \right\rceil = \left\lceil \frac{16,384}{16} \right\rceil = 2^{10} = 1024$ data block

Indexing:
One index record Size = Size of Key + Size of Pointer
= 22 + 10 = 32 Byte
Block factor of index file = $\left\lfloor \frac{\text{Block Size}}{\text{Index record Size}} \right\rfloor = \left\lfloor \frac{4096\ B}{32\ B} \right\rfloor$
$= 2^7$ Index entries / block.

Without index:

Ordered file:
To access a record avg no. of block access = $\lceil \log_2 B \rceil$
$= \lceil \log_2 1024 \rceil$
= 10 Block Access

Unordered file:
To access a record average no of block access = $\frac{B}{2} = \frac{1024}{2} = 512$ Block access

With Index:

Dense:
Total no. of index access entries = 16,384 $[2^{14}]$ (# DB records)
Bfr (Block factor of index file) = 128 Index record per block
Total no of index block = $\left\lceil \frac{16,384}{128} \right\rceil = 2^7$ Index block

Sparse:
Total #. of index entries = 1024 (# DB records)
Block factor of index file = 128 Index record per Block
Total no of index block = $\left\lceil \frac{1024}{128} \right\rceil = 2^3$ Index Block

Page No.
Date. / /

With Index: Dense.

To access a record index avg # block access = $\lceil \log_2 B \rceil$
$= \lceil \log_2 128 \rceil = 7$.

To access a record using index average # of block access = $\lceil \log_2 128 \rceil + 1$
$= 8$ Ans

Sparse:

To access index avg # block access $= \lceil \log_2 8 \rceil = 3$.

To access a record using index avg number of block access = $\lceil \log_2 Bi \rceil + 1$
$= \log_2 B + 1 = 4$ Ans

Meta Data (Data Dictionary)

→ record format
→ field format
→ number of blocks allocated for file
→ number of record in file.
→ Ordered field of file, etc.

Index File:

BF (Block Factor) of Index = $\left\lfloor \frac{B+H}{K+P} \right\rfloor$ entries/block.

Index file { | K_1 P_1 | K_2 P_2 | ⋮ ⋮ | }

$\therefore BF = \left\lfloor \frac{B+H}{K+P} \right\rfloor$

{# of DB file blocks} → {# of index blocks}

$\left\lfloor \frac{B+H}{R} \right\rfloor$ record/block $< \left\lfloor \frac{B+H}{K+P} \right\rfloor$ entries/block.

Types of Index

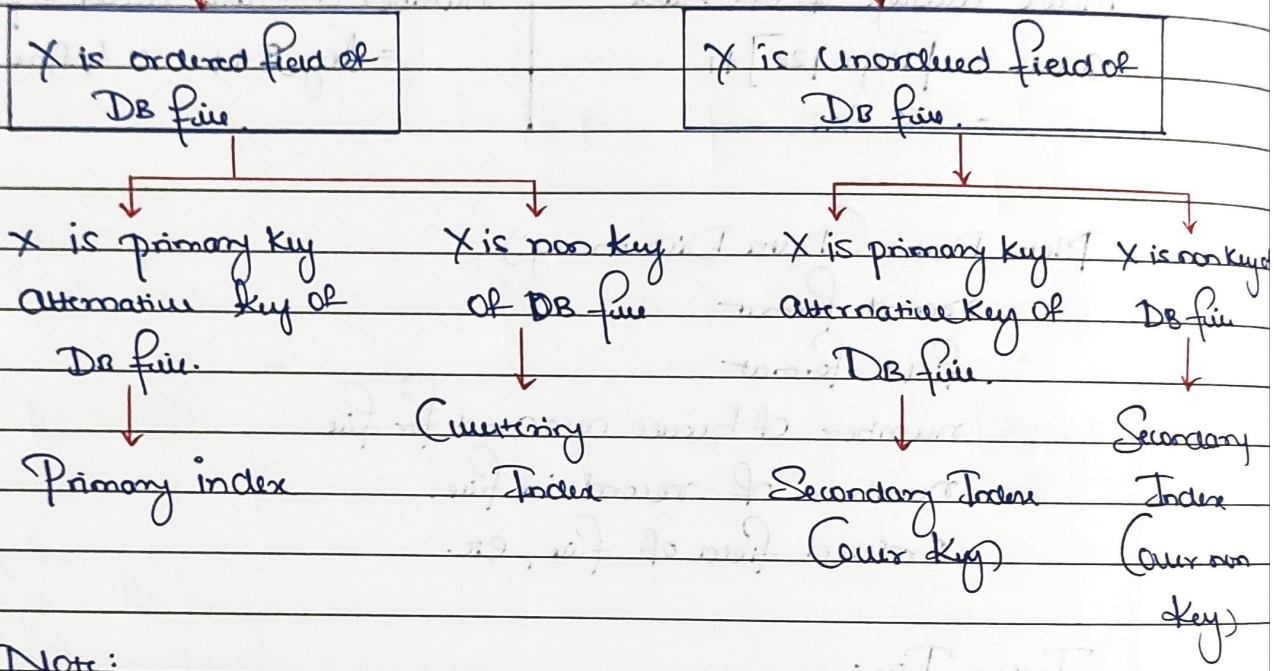
Single level Ordered Indexes.
1. Primary index
2. Clustering index
3. Secondary index

Multilevel Indexes
Dynamic multilevel indexes
Using B and B^+ tree.

Page No.
Date. 12/09/23

1. Primary index (key + ordered DB file)
2. Clustering index (non key + ordered file)
3. Secondary index (non key / key + unordered file)

Type of index.
(X: Search key)



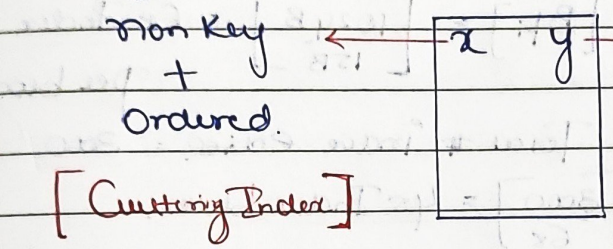
Note:
* At most one primary index possible per relation (Table).
* More than one secondary index possible.
* In a relation either CI or PI any one possible not both.

Primary Indexes:
* Ordered file with two fields:
  * Primary key, K(i)
  * Pointer to a disk block, P(i)
* One index entry in the index file for each block in the data file.
* Indexes maybe dense or sparse.
  * Dense index has an index entry for every search key value in the data file.
  * Sparse index has entries for only some search values.

Page No.
Date. / /

* Access cost to access record with PI with multilevel index : $(K+1)$ blocks.
* Primary index can be dense or sparse (Sparse is mostly used).
* For any database relation atmost one PI is possible (because of index over ordered field)

Q1. Data records based on non key (x) and index built over key field (y) of DB Table:



Q2. Data records ordered based on non-key and index build over same non key.

(non key + ordered.) ⟸ x

⟹ Clustering index

Q3 Suppose that we have ordered file of 30,000 records, stored on a disk with block size 1024 Byte (Record Size) file records are of fixed length and unspanned of size 100 Byte (Record size) and Suppose that we have created a primary index on the key field of the file of size 9 Byte and Block pointer of size 6 Byte then find the average number of Block Access to search for a record using with and without index?

Ans Records = 30,000 Block Size = 1024 B, Record Size = 100 B,
Key = 9B, Pointer = 6 Byte, [Ordered file, fixed length, primary Index]
Without-Index

Block factor of DB file = $\left\lfloor \frac{1024\ B}{100\ B} \right\rfloor$ = 10 Record/block

Page No.
Date. / /

Total number of DB record = 30,000
Total number of data block [B] = $\left\lceil \frac{30,000}{10} \right\rceil$ = 3000 Data Blocks

Ordered file: To access a record avg # of block access = $\lceil \log_2 B \rceil = \lceil \log_2 3000 \rceil$ = 12 Block access

With Index

One index record size = 9+6 = 15 Byte
Block factor of index file [BFi] = $\left\lfloor \frac{1024B}{15B} \right\rfloor$ = 68 Index Entries per block
Primary index: Sparse: Total # index entries = 3000 (# data block)
Total # index blocks = $\left\lceil \frac{3000}{68} \right\rceil$ = 45 Index blocks.

To access index block avg # of block access = $\lceil \log_2 Bi \rceil$
= $\lceil \log_2 45 \rceil$ = 6 Block access
To access record using index block, avg # of block access
= $\lceil \log_2 45 \rceil + 1$ = 7 block access.

Clustering Indexes

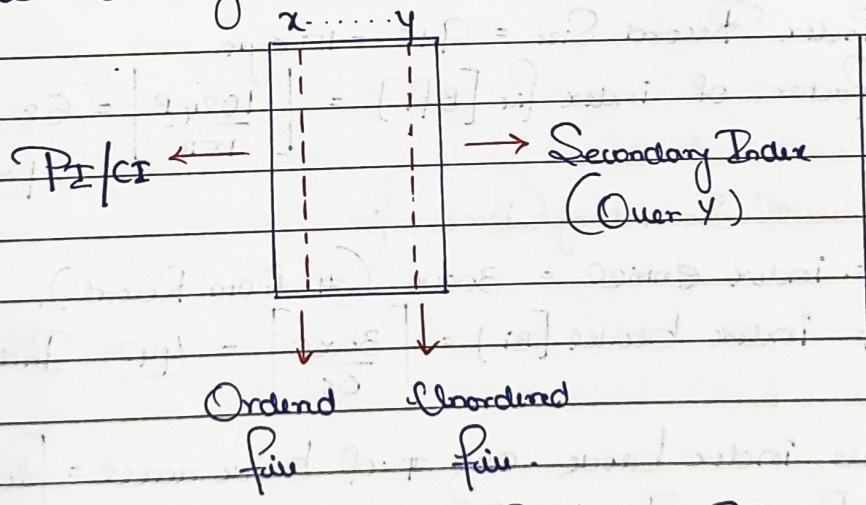
* Clustering field
  * File records are physically ordered on a non key field without a distinct value for each record.
* Ordered file with two fields
  * Same type as Clustering field
  * Disk block pointer.
* Clustering index mostly Sparse index (Dense CI also possible if each cluster with one record).
* At most one CI is possible for any database relations (ordering required)
* For any DB relation can build either PI or CI but not both.

Page No.
Date. 13/09/23

Secondary Indexes

* A Secondary index provides a secondary means of accessing a file for which some primary access already exists.
* The secondary index may be on a field which is a candidate key and has a unique value in every record, or a non key with duplicate values.
* The index is an ordered file with two fields:
  1. The first field is of the same type (data type) as some non ordering field of that data file is an indexing field.
  2. The secondary field is either a block pointer or a record pointer. There may be many secondary indexes (and hence indexing fields) for the same file.
* Includes one entry for each record in the data file, hence it is a dense index.

→ Search Key: Unordered field and key/Non key
→ Secondary possible way to access data using index even PI/CI indexes already exists.



More than one SI possible for any database table : No ordered table required

→ I/O cost to access record using SI over key with MLI is (K+1) blocks
→ I/O cost to access record of some non-key using SI over non key with MLI : { K + # of blocks of DB equal to # of pointers in given redirect page }.

Q4. Consider a Secondary index on the key field of the file of question number 1, then find the average number of block access to access a record using with and without index?

Ans. #. Records = 30,000, Block Size = 1024 B, Record Size = 100 B
Key = 9B, Pointer = 6B.
{ unordered file. Secondary (Dense Index) }

Without Index

Block factor of DB file (Unspanned) $= \left\lfloor \frac{1024B}{100B} \right\rfloor$ = 10 Record/block.

Total number of DB record = 30,000.

Total number of Data Block [B] $= \left\lceil \frac{30000}{10} \right\rceil$ = 3000 Data Blocks

Ordered file

To access a record avg #. block access $= \frac{B}{2} = \frac{3000}{2}$ = 1500 Block access

With Index

One Index Record Size = 9+6 = 15 Byte

Block factor of index file [Bfi] $= \left\lfloor \frac{1024B}{15B} \right\rfloor$ = 68 Index Entries Per block.

{ Unordered, Secondary (Dense) }

Total #. index entries = 30000 (#. Data Record).

Total #. index blocks [Bi] $= \left\lceil \frac{30000}{68} \right\rceil$ = 442 Index block.

To access index block avg #. of block access $= \lceil \log_2 Bi \rceil$
$= \lceil \log_2 442 \rceil$ = 9 Block access

To access record using index avg # block access $= \lceil \log_2 Bi \rceil + 1$
$= \lceil \log_2 442 \rceil + 1$ = 10 Block access

Page No.
Date. / /

Multilevel Index

* If index does not fit in memory, access becomes expensive
* Solution: Treat index kept on index as a sequential file and construct a sparse index on it.
  - → Outer index: a sparse index of the basic index
  - → Inner index: the basic index file.
* If even outer index is too large to fit into main memory, yet another level of index can be created and so on.
* Indices at all levels must be updated on insertion or deletion from the file.

→ Multilevel indexes are designed to greatly reduce remaining search space as search is conducted.
→ Index file: Considered first (or base level) of a multilevel index
→ Second level: Primary index to the first level
→ Third level: Primary index to the second level.

Note: * We can repeat the above process until index entries fit into one block.
* If there are n level in multilevel index then the number of block access to search for a record = n+1
(for each level one index block + 1 data block).

Qs. Find the average number of block access required to search for a record if multi level index is created on the data file of previous question.

Ans Block factor of Index file = 68 entries per block
I Level: Total no. of index block = 442 index block.
II level: no. of index records (entries) = 442 (Sparse no. of 1st level block) and Block factor = 68 entries for block.
Total number index block = $\lceil \frac{442}{68} \rceil$ = 7 index blocks.

III Level: Number of index record = 7 (number of 2nd level block)
Total number of index block = $\lceil \frac{7}{68} \rceil = 1$

Average no. of block access = 1+1+1+1 = 4.

Q6 A clustering index is defined on the fields which are of type:
Ans Non Key and Ordering.

Q7. Consider a file. A file of 16384 records. Each record is 32 byte long and its key field is of size 6 bytes. The file is ordered on a non key field, and the file organization is unspanned. The file is stored in a file system with block size 1024 bytes, & size of block pointer is 10 byte. If the secondary index is built on the key field of the file, and a multi level index scheme is used to store the secondary index, the number of first-level and second-level blocks in the multi level index are respectively are:

Ans #. Records = 16,384 [2^{14}] Record Size = 32 B, Key = 6B, BP = 10 B.
Block Size = 1024 B, Secondary Index (Dense) & multi level index:
One index record size = 6+10 = 16 Byte
Block factor of Index file [BFi] = $\lfloor \frac{1024B}{16B} \rfloor = \frac{2^{10}}{2^4} = 2^6 = 64$ Index entries Per block.

Secondary Index: (Dense).
Total #. index entries = 16,384 (# DB records)
I Level: Total # index block = $\frac{2^{14}(16,384)}{2^6(64)} = 2^8 = 256$ Index Block

II level: #. index entries = 256 (# Ist level blocks)
BFi = 64 Index entries per block
Total # index blocks = $\lceil \frac{256}{64} \rceil = 4$

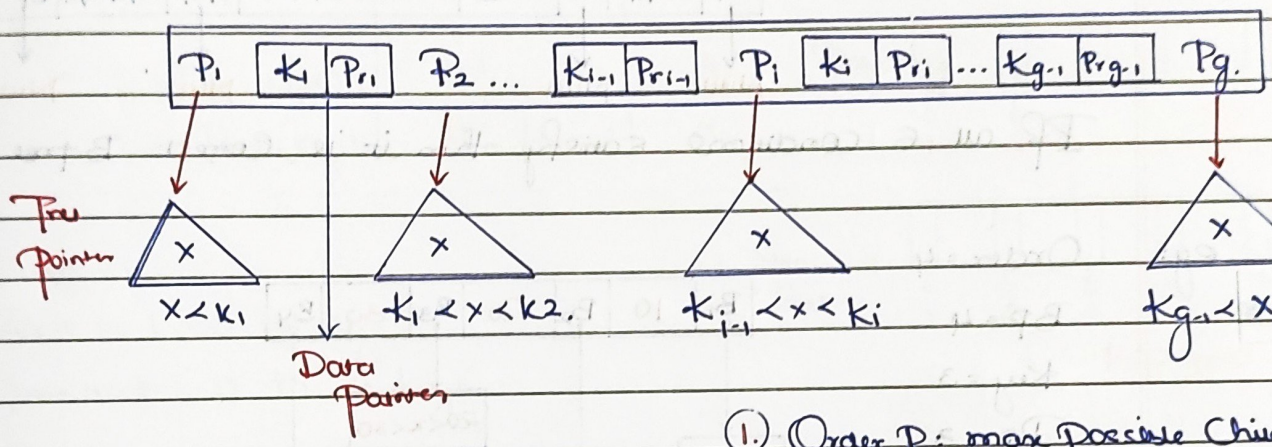
∴ Ans = {256, 4}

Page No.
Date. / /

III Level : Total # index entries = 4, & BFi = 64 Index entries per block.
Total # index blocks = $\lceil \frac{4}{64} \rceil = 1$

∴ So total 3 level required.

B Tree Structure :



P: Block/Child/Index Pointer
Pr: Read/Record/Data Pointer
K: Search Key

① Order P: max possible Child Pointers [degree] can store in B Tree node

B-Tree Definition

② Every internal node except the root node contain atleast (min) $\lceil P/2 \rceil$ block pointer (min keys $\lceil P/2 \rceil - 1$) and maximum P block pointer (maximum keys P-1).

③ Root can contain atleast 2 block pointer (min 1 key in root node) and maximum P block pointer (max P-1 keys).

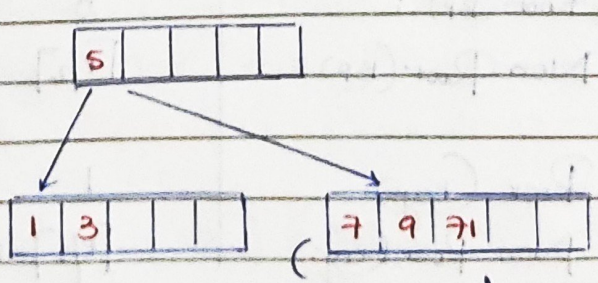
④ Keys within the node should be in ascending order.

⑤ Each leaf node should be at same level.

eg. Order = 6 → 1, 5, 3, 7, 9, 11, 13, 16 → | 1 | 3 | 5 | 7 | 9 |

Min key = $\lceil 6/2 \rceil - 1 = 2$

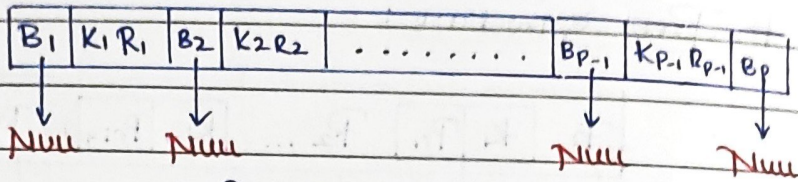
Maximum keys = 6-1 = 5.



Order = P
BP = P
key = P-1
RP = P-1

- Order : P
- Max Keys = P-1
- Min Keys = $\lceil P/2 \rceil - 1$
- Max Bp = P
- Min. Bp = $\lceil P/2 \rceil$

⑥ Structure of leaf node:



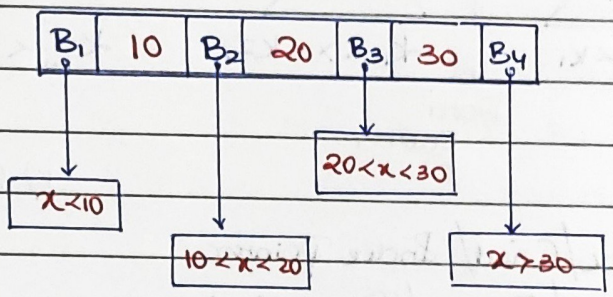
If all 6 conditions satisfy then it is Correct B+tree of order P.

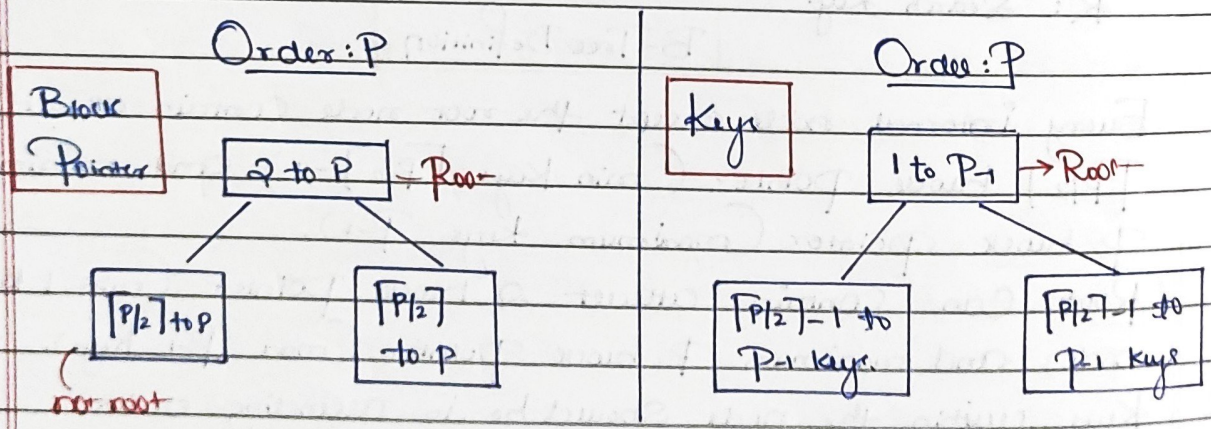
eg: Order = 4

Bp = 4

Key = 3

Rp = 3.





Order : P

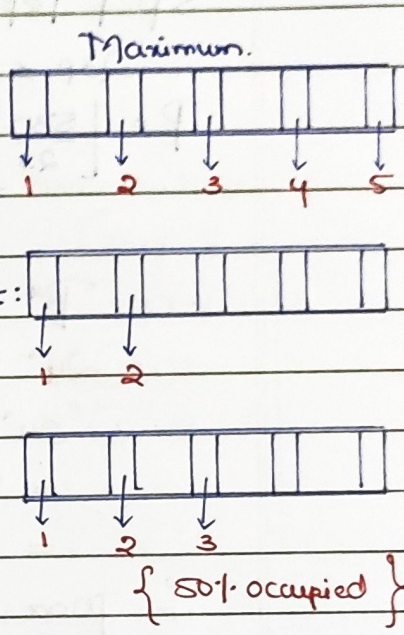
	Minimum	Maximum
Root (Bp) :	2	P
Non Root (Bp) :	$\lceil P/2 \rceil$	P
Root (keys)	1	P-1
Non Root (keys)	$\lceil P/2 \rceil - 1$	P-1

Page No.
Date. 14/09/23

Note: Root cannot have only one block pointer.

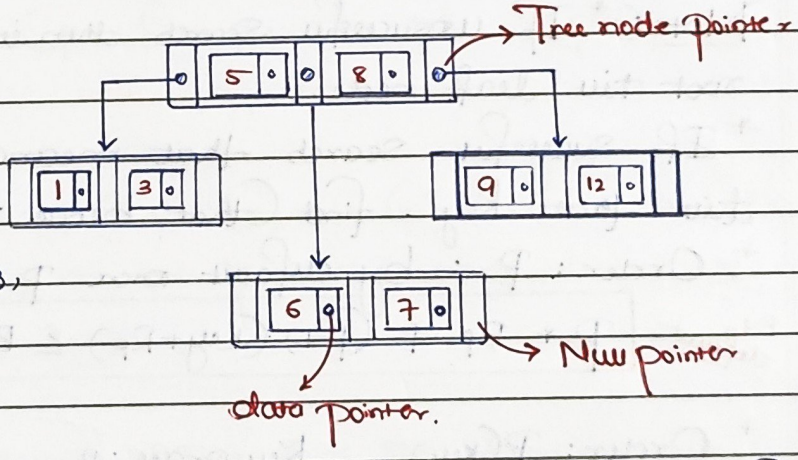
Order: P [5].

	Maximum.	Minimum.
Root (BP)	2	5
Non Root (BP)	$\lceil 5/2 \rceil = 3$	5
Root (Keys)	1.	4
Non Root (keys)	$\lceil 5/2 \rceil - 1 = 2$	4.



Order: 5
Root: 1 to 4 keys
Non root: 2 to 4 keys

→ A B-Tree of order P=3. The values were inserted in the order 8, 5, 1, 7, 3, 12, 9, 6.



Q1. Consider a table in a relational database with a key field K. A B tree of order p is used as an access structure on K, where p denotes the maximum number of tree pointers in B-tree index node. Assume that K is 10-byte long, disk block size is 512 byte. Each data pointer size Pb is 8 byte long and each block pointer Pp is 5 byte long. To order for each B-tree node to fit into a single disk block, the maximum value of p is

Ans Key = 10 Byte, Pb = 8B, Bp = 5B. Block Size = 512 Byte
Order : P.

$$P * B_p + (P-1)[Key + P_b] \leq \text{Block Size}.$$

Page No.
Date. / /

$P*5 + (P-1)[10+8] \leq 512$
$5P + (P-1)[18] \leq 512$
$5P + 18P - 18 \leq 512$
$23P \leq 530$
$P = \left\lfloor \frac{530}{23} \right\rfloor = \lfloor 23.04 \rfloor = \underline{23}$

If we take 24 then we get the values of order ($P=24$) then it exceeds by block size 512 Byte
Or
It cannot accomodate in a block of size 512 Byte.

$P*Bp + (P-1)(Key+Rp)$
$24*Bp + (24-1)(Key+RP)$ } > Block Size (512 Byte)
$24*5 + 23(18)$
$= 120 + 414$
$= 534$ Byte, But given block size is 512 Byte.
∴ Max order is 23 not 24.

Note: * If unsuccessful search then it means we traversed from root till leaf node.
* If successful search that means we traverse from root to that key. find (that block in which key is present).
* Order: P : by default max P block pointer

By default: $P * Bp + (P-1)(key+Rp) \leq$ Block size.

* Order: P(key), key order: p

$(p+1)Bp + P(Key_p + Rp) \leq$ Block size

Order: P Minimum

Height/level	Min # nodes	Min # Bp	Min # keys
0/1	1	2	1
1/2	2	$2\lceil P/2 \rceil$	$2(\lceil P/2 \rceil - 1)$
2/3	$2\lceil P/2 \rceil$	$2\lceil P/2 \rceil^2$	$2\lceil P/2 \rceil(\lceil P/2 \rceil - 1)$
⋮	⋮	⋮	⋮
h/h+1	$2\lceil P/2 \rceil^{h-1}$	$2\lceil P/2 \rceil^{h}$	$2\lceil P/2 \rceil^{h-1}(\lceil P/2 \rceil - 1)$

* level = height + 1

Page No.
Date. / /

→ Total Minimum # Keys = $\left[1 + 2(\lceil p/2 \rceil - 1) + 2\lceil p/2 \rceil(\lceil p/2 \rceil - 1) + \cdots 2\lceil p/2 \rceil^{h-1}(\lceil p/2 \rceil - 1)\right]$ → GP Formula.

Order : P Maximum.

Height / Level.	Max # Nodes	Max # BP	Maximum Keys
0/1	1	P	(P-1)
1/2	P	P^2	P(P-1)
2/3	P^2	P^3	$P^2(P-1)$
3/4	P^3	P^4	$P^3(P-1)$
⋮	⋮	⋮	⋮
h/h+1	P^h	P^{h+1}	$P^h(P-1)$

Total Maximum # Keys $= (P-1) + P(P-1) + P^2(P-1) + P^3(P-1) \cdots + P^h(P-1)$
$= (P-1)(1 + P + P^2 + P^3 \cdots P^h)$.
⟹ Apply GP Series

Q1. Order : 5
Level : 5
$= (5-1) + (5\times4) + (5^2\times4) + (5^3\times4) + (5^4\times4)$
$= 4 + 20 + 100 + 500 + 2500 \Longrightarrow 3124$
Total max no. of keys = 3124.

Q2 Order : 23 : Consider a BTree 69% Full, then minimum number of keys = ?

Ans No. of BP per node = 23×0.69
$= 16$
Fanout = 16

(P) Order : 16
(P-1) Keys = 15.

Note: Maximum level : At each level min # key & min # BP
$(\lceil P/2 \rceil - 1)$ $\lceil P/2 \rceil$

Minimum level : At each level maximum # key & max # BP
(P-1) (P)

{ Applicable for both B and B+ Tree }

Page No.
Date. / /

Advantage of B-Tree:

1. B Tree index best suitable for random access of some record.

```
Select *
From R
Where A = 24;
```

(One record access.)

I/O Cost : K+1 blocks
$= [\log_p n] + 1 = O(\log_p n)$

Disadvantage of B-Tree:

1. B Tree index not best suitable for sequence access of range of records.

```
SELECT * FROM R
WHERE A ≥ 30 and A ≤ 85;
```

(Range of records)

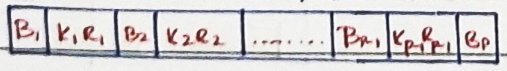
I/O Cost : $x(\log_p n + 1)$ + cost of unsuccessful more access cost (Unordered file DB).

B+ Trees

1. All keys are available at leaf node
2. Internal node: no read pointer
3. Leaf node contain key & R_p + 1 Block Pointer.

Difference between BTree and B+ Tree

1. BTree Internal Node:

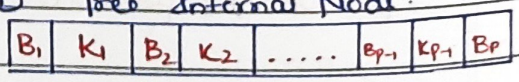


Order : p (maximum # Bp)

$B_p = P$, $keys = P-1$, $R_p = P-1$.

$P * B_p + (P-1)(key + R_p) \leq$ Block Size

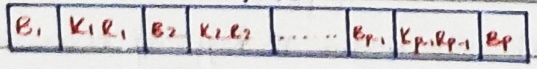
B+ Tree Internal Node:



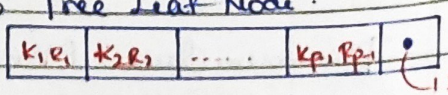
Order : p (maximum # pointer)

$P * B_p + (P-1) key \leq$ Block Size

2. BTree leaf Node:



B+ Tree leaf Node:



(1 Block pointer)

Page No.
Date. / /

Order : p (max #. of Bp)
Bp = P, Key = P-1, Rp = P-1

$$P * Bp + (P-1)(Key + Rp) \leq \text{Block Size}$$

Order : p (maximum # pointers)

$$(p-1)(Key + Rp) + 1Bp \leq \text{Block Size}$$

Note: * In a B Tree internal node and leaf node have same structure (order of internal node is same as order of leaf node in B tree)
* In a B+ Tree internal node and leaf node have different structure (order of internal node is not same as order of leaf node).

eg:1 Block size = 1024 Byte, Key = 6 Byte, Bp = 8B, Rp = 10 Byte order of internal node and leaf node in a B Tree?

$$P * Bp + (P-1)(Key + Rp) \leq \text{Block Size}$$

$P * 8 + (P-1)(6+10) \leq 1024$
$8P + 16P - 16 \leq 1024$
$P \leq 24 \leq 1024 \Rightarrow \left\lfloor \frac{1024}{24} \right\rfloor = 43.$

$\therefore P = 43$

Order
Internal node : 43
Leaf node : 43

eg:2 Consider block size = 1024 Byte, Key = 6 Byte, Bp = 8B, Rp = 10 Byte
Order of internal and leaf node in a B+ Tree.

Order of internal node

$$P * Bp + (P-1) Key \leq \text{Block Size}$$

$P * 8 + (P-1) 6 \leq 1024$
$8P + 6P - 6 \leq 1024$
$14P \leq 1030$
$P = \left\lfloor \frac{1030}{14} \right\rfloor = 73.$

Order of leaf node in B+ Tree.

$$(P-1)(Key + Rp) + 1Bp \leq \text{Block Size}$$

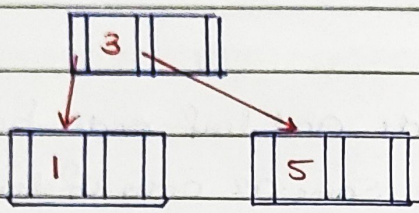
$(P-1)(6+10) + 8 \leq 1024$
$16P - 16 + 8 \leq 1024$
$16P \leq 1032$
$P = \left\lfloor \frac{1032}{16} \right\rfloor = 64$

Page No.
Date. / /

eg: Keys = 1, 3, 5.

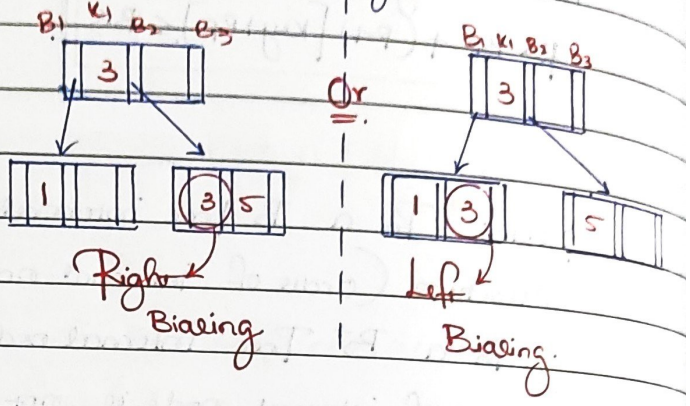
B Tree

Order : 3 max keys = 2.

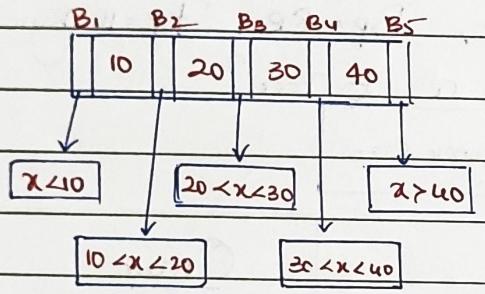


B+ Tree

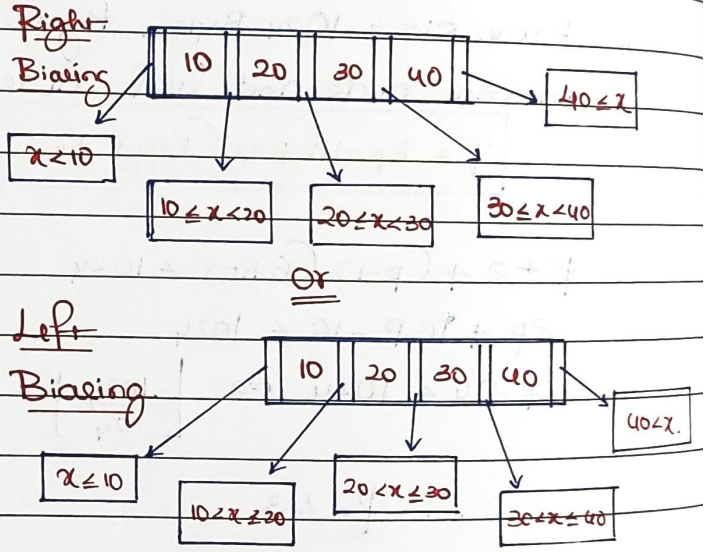
Order : 3, Max Keys = 3.



eg B Tree : Order : 5

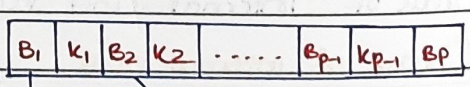


B+ Tree Order : 5



B+ Tree Definition

1. Structure of Internal node:



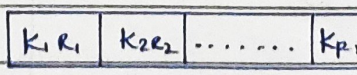
Left biasing: => $x \le K_1$ | $K_1 < x < K_2$

Or

Right biasing => $x < K_1$ | $K_1 \le x < K_2$

$P * B_P + (P-1) Key \le$ Block Size.

2. Structure of leaf node:



$(P-1)[Key + R_P] + 1 B_P \le$ Block Size

* Remaining 4 points are same as B tree

Page No.
Date. / /

Q1. Why internal node not having Rp?

Ans As all keys are available at leaf node so read pointer not required in internal node.

Q2 Why leaf node has only key and Rp only?

Ans Because all the keys are available in leaf node according to B+ definition so to read those keys (data) read/data pointer is required.

Q3. Why leaf node having only 1 block pointer?

Ans B+ Tree is suitable for range query, and in B+ tree all keys are available in the leaf node and if 1 block connect to next node at leaf (like linked list) so I/O cost is reduced while performing range query.

Q4. When all keys are available at leaf node then why some keys are available in non leaf node?

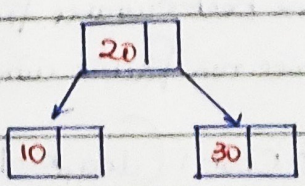
Ans Yes there are some keys available in the non leaf node but they are very useful in searching purpose. It is only used for searching purpose, they don't have Real (Record/data) pointer in non leaf node.

Creation/Insertion in B+ Tree

1. In B+ Tree insertion start from leaf node.
2. Split of internal node is same as split of internal node in B Tree.

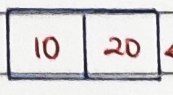
Internal Node

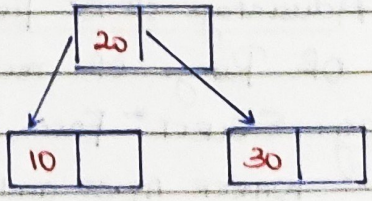
B Tree:



Internal Node

B+ Tree:



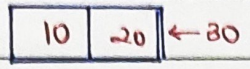


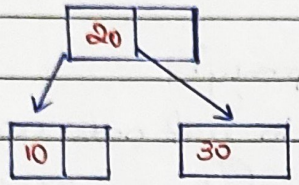
Page No.
Date. / /

③ Split of leaf in B^+ Tree is different from B Tree.

Leaf Node:

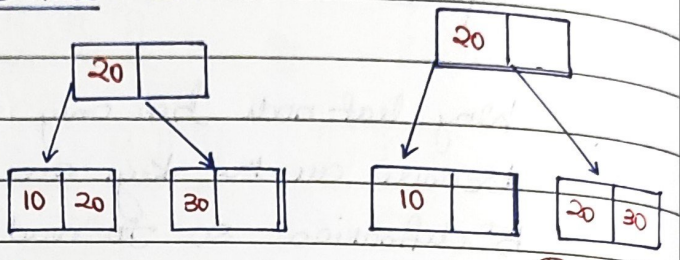
B Tree.





Leaf Node:

B^+ Tree

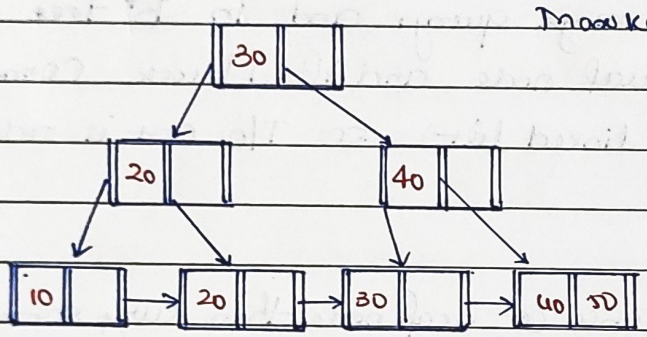


Left Biasing.

Right Biasing

eg: Keys: 10, 20, 30, 40, 50. Order: 3 {Right Biasing}

Max key = 2



Important Points:

* Every key of leaf node is not present in non leaf node.
  example: Keys = 10, 20, 30, 40, 50 in leaf node but all key are not present in non-leaf node (except 10, 50 all are present in previous example).
* Some key of leaf node is present in non leaf node.
* But every key of non leaf node must be present in leaf node (i.e in B^+ Tree all keys are available at the leaf node).
* In B^+ Tree: leaf-node key order:

$$P(key + Rp) + 1Bp \leq \text{Block Size}$$

Advantage of B^+ Tree: B^+ Tree index best suitable for sequence access of Range of records. (Range Queries run faster with B^+ Tree index).

eg: SELECT * FROM R
WHERE A ≥ 30 and A < 85; → I/O Cost: $(\log_{p^n} + x + z)$ (x for index, z for DB file)