Introduction to Data Structure (Data Management) Lecture 13

Felipe P. Vista IV

Reminder

- Everybody, make sure that your name in ZOOM is in the following format:
  - University ID Num Name (no “(   )“)
  - Ex: 202054321 Juan Dela Cruz
  -
  - Not changing your name to this format
    - you might be marked Absent

  * → absent?

- Data Storage Basics

- Indexes

INTRO TO DATA STRUCTURE

Data Storage Basics

Motivation

- To understand performance, we need to understand a bit about how DBMS works:
  - the database application is too slow… why?
  - one of the queries is very slow… why?

Motivation

- To understand performance, we need to understand a bit about how DBMS works:
  - the database application is too slow… why?
  - one of the queries is very slow… why?
- Understanding query optimization
  - we've seen SQL query ~> logical plan (RA),
    but not much about RA ~> physical plan

Motivation

- To understand performance, we need to understand a bit about how DBMS works:
  - the database application is too slow… why?
  - one of the queries is very slow… why?
- Understanding query optimization
  - we've seen SQL query ~> logical plan (RA),
    but not much about RA ~> physical plan
- Choice of indexes is often up to us

Student

Id	fName	lName
10	Matt	Burt
20	Makenna	Balvanz
…	…	…

Data Storage

- DBMSs store data in files

Student

Id	fName	lName
10	Matt	Burt
20	Makenna	Balvanz
…	…	…

Data Storage

- DBMSs store data in files
- Most common organization is row-wise storage:
  - file is split into blocks
  - Each block contains a set of tuples

10	Matt	Burt	block 01
20	Makenna	Balvanz	
50	…	…	block 02
200	…		
220			block 03
240			
420			block 04
800			

The given example has 4 blocks with 2 tuples each

Student

Id	fName	lName
10	Matt	Burt
20	Makenna	Balvanz
…	…	…

Data Storage

- DBMSs store data in files
- Most common organization is row-wise storage:
  - file is split into blocks
  - Each block contains a set of tuples
- DBMS reads the entire block

10	Matt	Burt	block 01
20	Makenna	Balvanz	
50	…	…	block 02
200	…		
220			block 03
240			
420			block 04
800			

The given example has 4 blocks with 2 tuples each

Student

Id	fName	lName
10	Soheill	Satavis
20	Nwabisa	Ngumbela
…	…	…

Data File Types

The data heap file can be one of:

- Heap file
  - unsorted

Student

Id	fName	lName
10	Soheill	Satavis
20	Nwabisa	Ngumbela
…	…	…

Data File Types

The data heap file can be one of:

- Heap file
  - unsorted
- Sequential file
  - sorted based on some attributes

Student

Id	fName	lName
10	Soheill	Satavis
20	Nwabisa	Ngumbela
…	…	…

Data File Types

The data heap file can be one of:

- Heap file
  - unsorted
- Sequential file
  - sorted based on some attributes

Note:
The key here is something different from primary key:
- it just means that we order the file according to that attribute.

In our example, we order by ID.
- we can also order by fName,
--- if it seems a better idea for applications using the DB.

Index

- An additional file, that allows fast access to records in the data file given a search key

Index

- An additional file, that allows fast access to records in the data file given a search key
- The index contains (key, value) pairs:
  - Key:= attribute value (e.g. Student ID, name)
  - Value:= pointer to the record

Index

- An additional file, that allows fast access to records in the data file given a search key
- The index contains (key, value) pairs:
  - Key:= attribute value (e.g. Student ID, name)
  - Value:= pointer to the record
- Could have many indexes for one table
  - sorted based on some attributes

Key = it means as a search key

What Key?

Different **keys**:

- Primary key – uniquely identifies a tuple

What Key?

Different **keys**:

- Primary key – uniquely identifies a tuple
- Key of Sequential file – how data file is sorted, if at all

What Key?

Different **keys**:

- Primary key – uniquely identifies a tuple
- Key of Sequential file – how data file is sorted, if at all
- Index Key – how index is organized

Ex. 1: Index on ID

Student

Id	fName	lName
10	Soheill	Satavis
20	Nwabisa	Ngumbela
…	…	…

Index on Student.ID

10	
20	
50	
200	
220	
240	
420	
800	

950	
…	

Student

Id	fName	lName
10	Soheill	Satavis
20	Nwabisa	Ngumbela
...	...	...

Ex. 1: Index on ID

Index on **Student.ID**

10	
20	
50	
200	
220	
240	
420	
800	
950	
...	

Data file **Student**

10	Soheil	Satavis
20	Nwabisa	Ngumbela
50	...	...
200	...	
220		
240		
420		
800		

Ex. 1: Index on ID

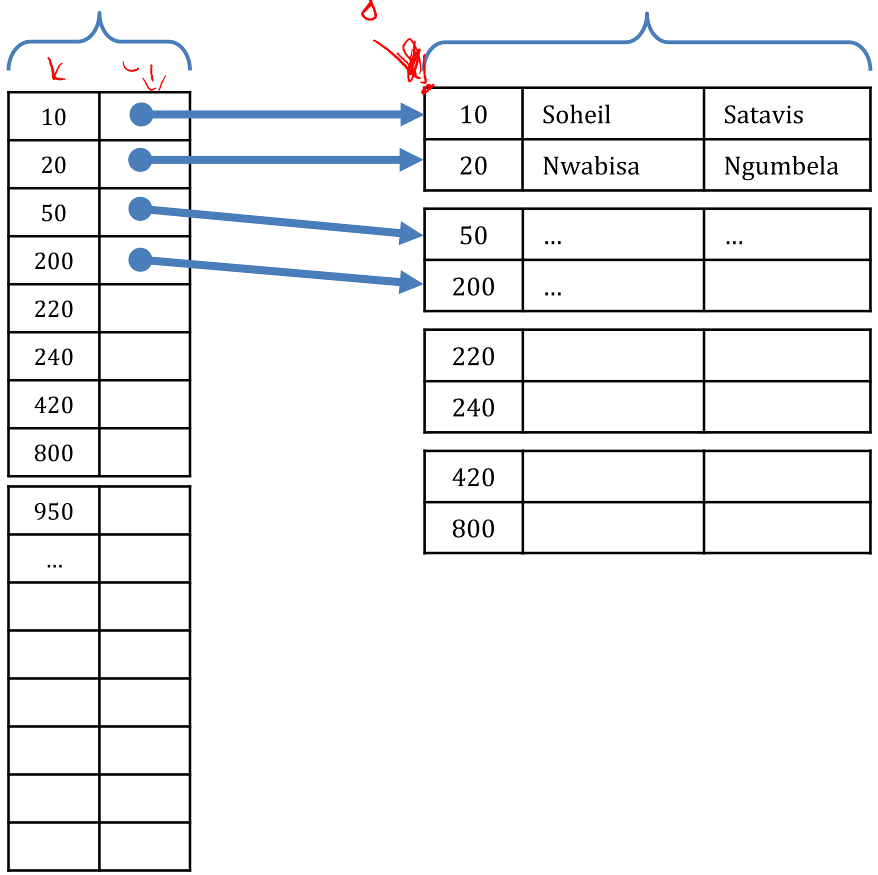
Student

Id	fName	lName
10	Soheill	Satavis
20	Nwabisa	Ngumbela
…	…	…

File

Index on Student.ID

Data file Student



Ex. 2: Index on fName

Student

Id	fName	lName
10	Soheill	Satavis
20	Nwabisa	Ngumbela
…	…	…

Index on Student.fName

Nwabisa	
Pat	
Janin	
Mikki	
…	
…	
…	
…	

…	
…	
Soheil	

Student

Id	fName	lName
10	Soheill	Satavis
20	Nwabisa	Ngumbela
…	…	…

Ex. 2: Index on fName

Index on Student.fName

Nwabisa	
Pat	
Janin	
Mikki	
…	
…	
…	
…	

…	
…	
Soheil	

Data file Student

8 Nwabisa

10	Soheil	Satavis
20	Nwabisa	Ngumbela

50	…	…
200	…	

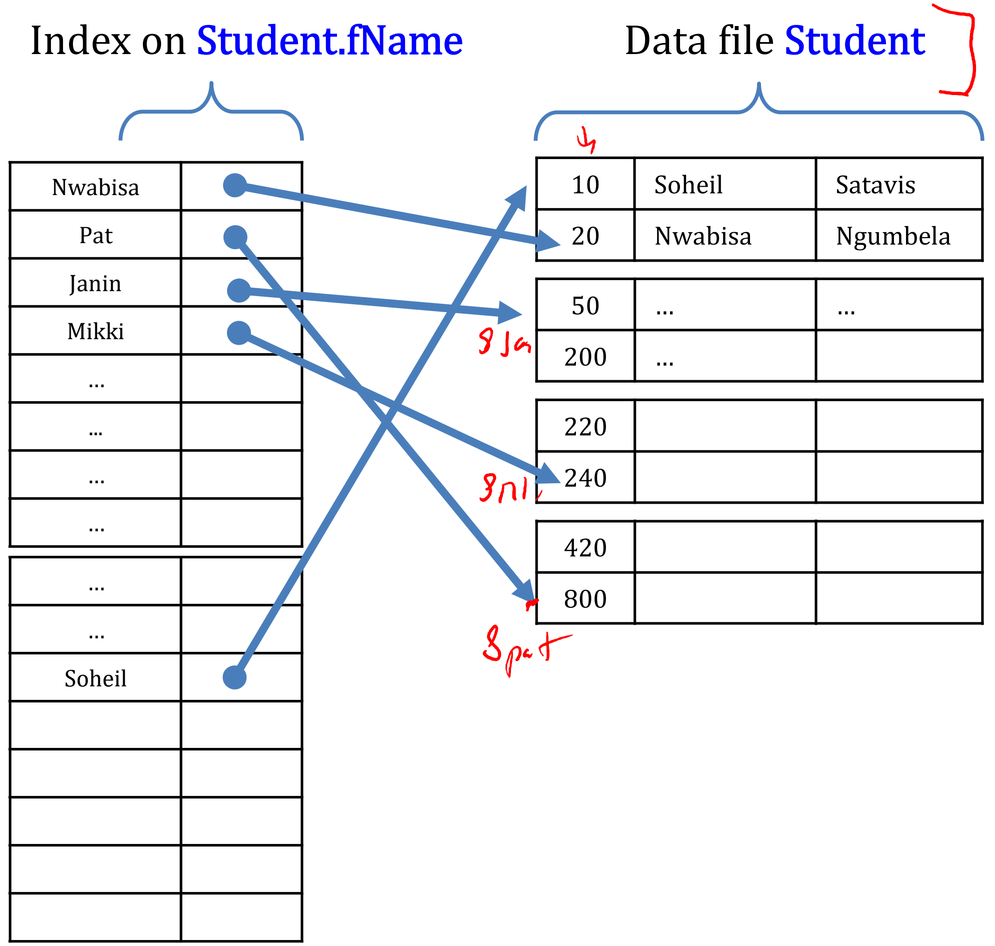
220		
240		

420		
800		

Ex. 2: Index on fName

Student

Id	fName	lName
10	Soheill	Satavis
20	Nwabisa	Ngumbela
…	…	…



Index on lName

Pelyo	8 Pat
Gates	8 Sdan
Balvant	8 Mic

Index on Organization

Several index organizations:

- B+ trees – most popular
  - They are search trees,
  - but they are not binary, instead have higher fan-out

Index on Organization

Several index organizations:

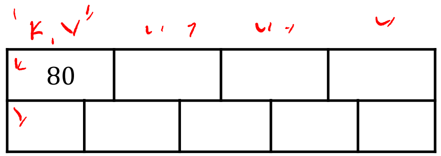
- B+ trees – most popular
  - They are search trees,
  - but they are not binary, instead have higher fan-out
- Hash table

Index on Organization

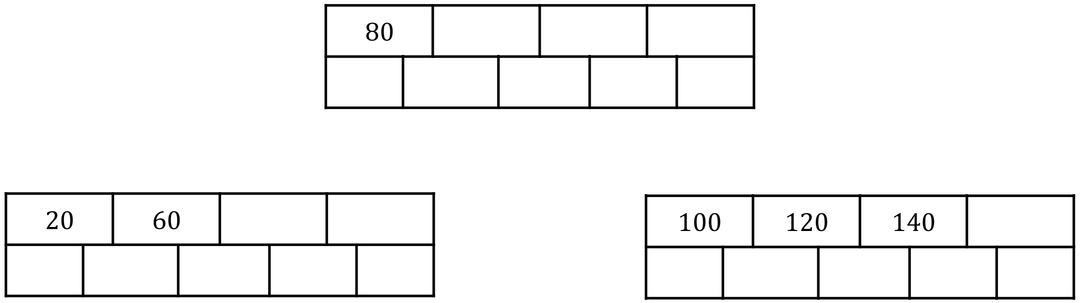
Several index organizations:

- B+ trees – most popular
  - They are search trees,
  - but they are not binary, instead have higher fan-out
- Hash table
- Specialized indexes – bit maps, R-trees, inverted index

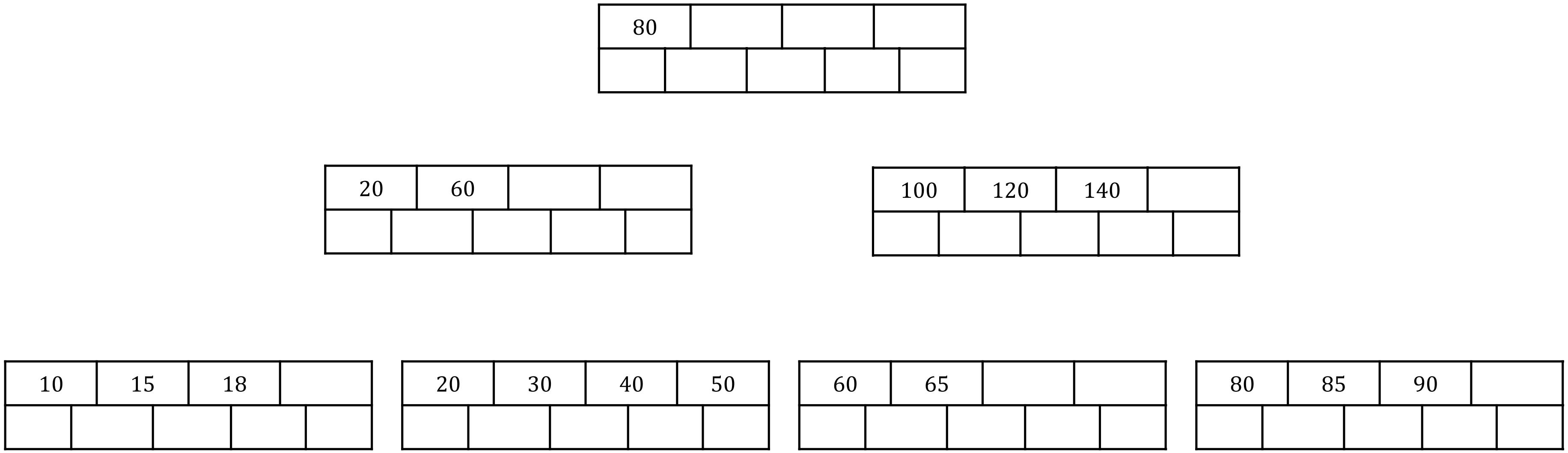
Recap: B+ Tree



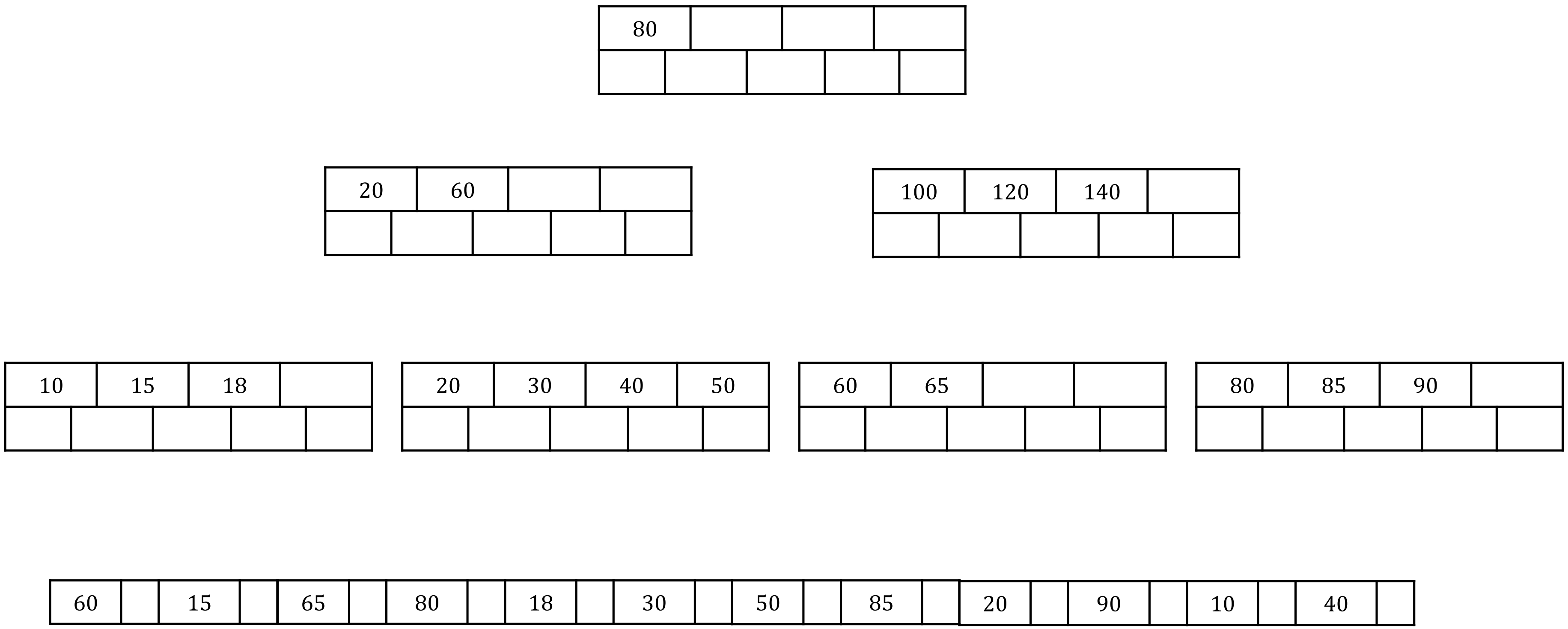
Recap: B+ Tree



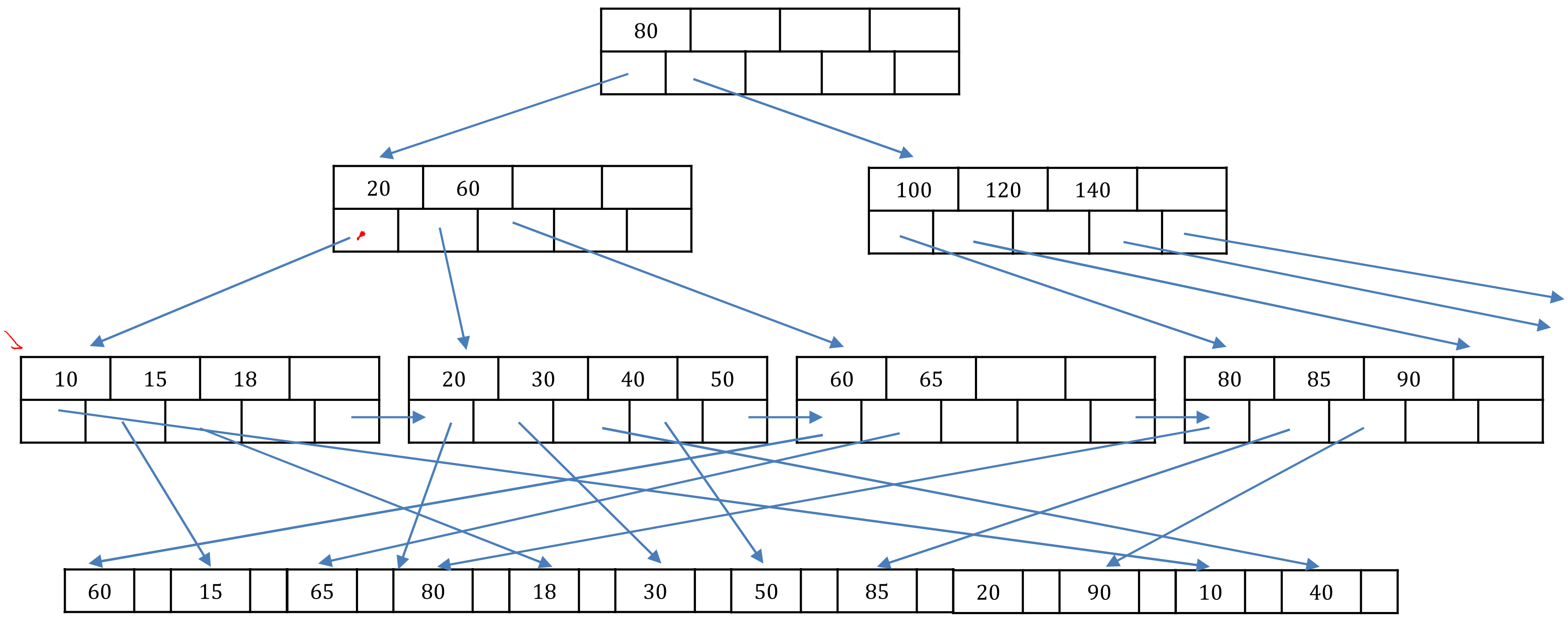
Recap: B+ Tree



Recap: B+ Tree


80
20 60
100 120 140
10 15 18
20 30 40 50
60 65
80 85 90
60 15 65 80 18 30 50 85 20 90 10 40

Recap: B+ Tree


80
20
60
100
120
140
10
15
18
20
30
40
50
60
65
80
85
90
60
15
65
80
18
30
50
85
20
90
10
40

www.jbnu.ac.kr

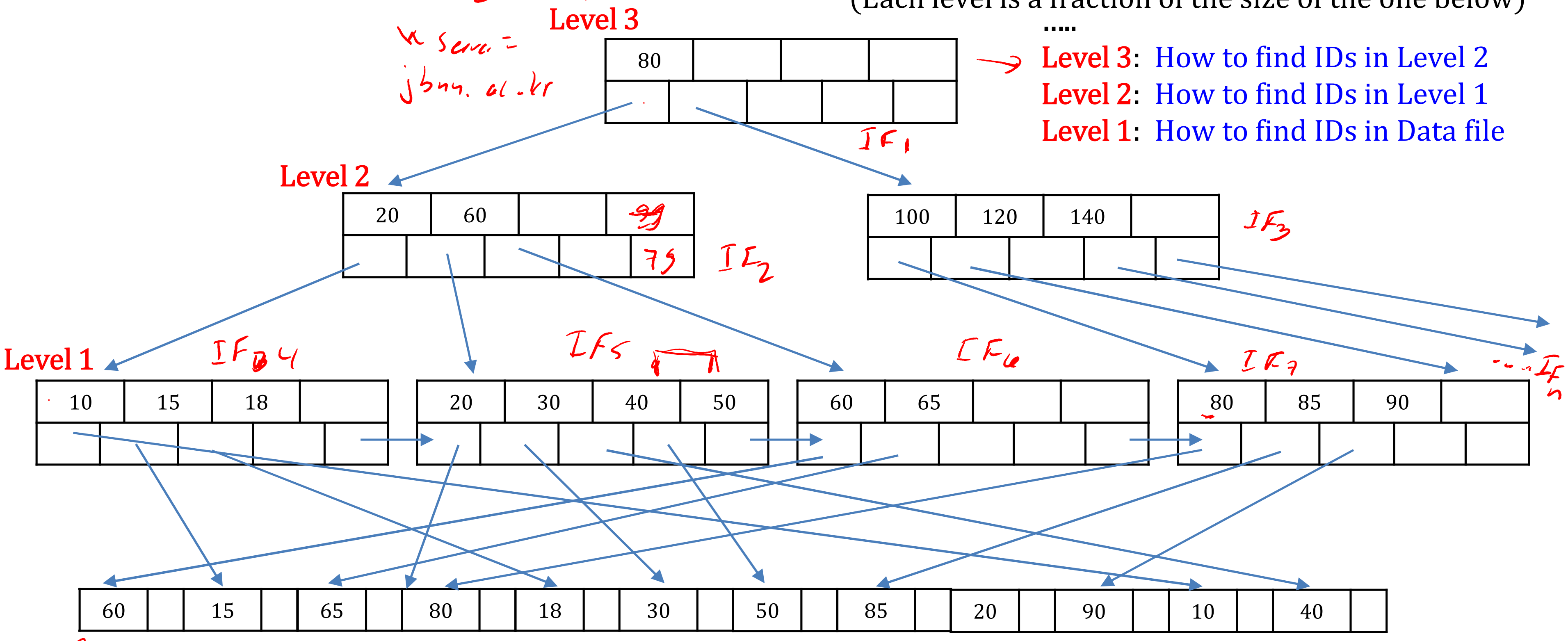
DNS 1 = .kr
DNS 2 = .ac.kr
ac server = jbnu.ac.kr

Recap: B+ Tree

(Each level is a fraction of the size of the one below)

.....

Level 3: How to find IDs in Level 2
Level 2: How to find IDs in Level 1
Level 1: How to find IDs in Data file



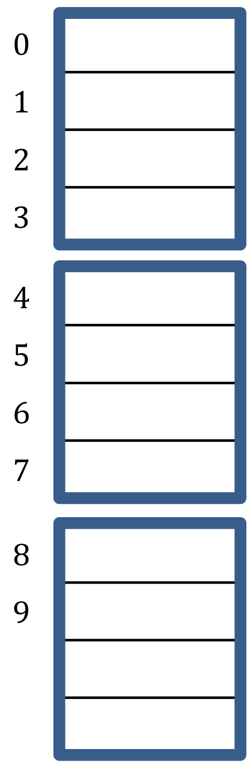
id dB File

Hash Index

A (naïve) hash function:

$$\boxed{H(x) = x \bmod 10}$$

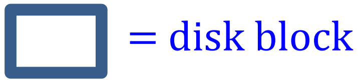
 = disk block

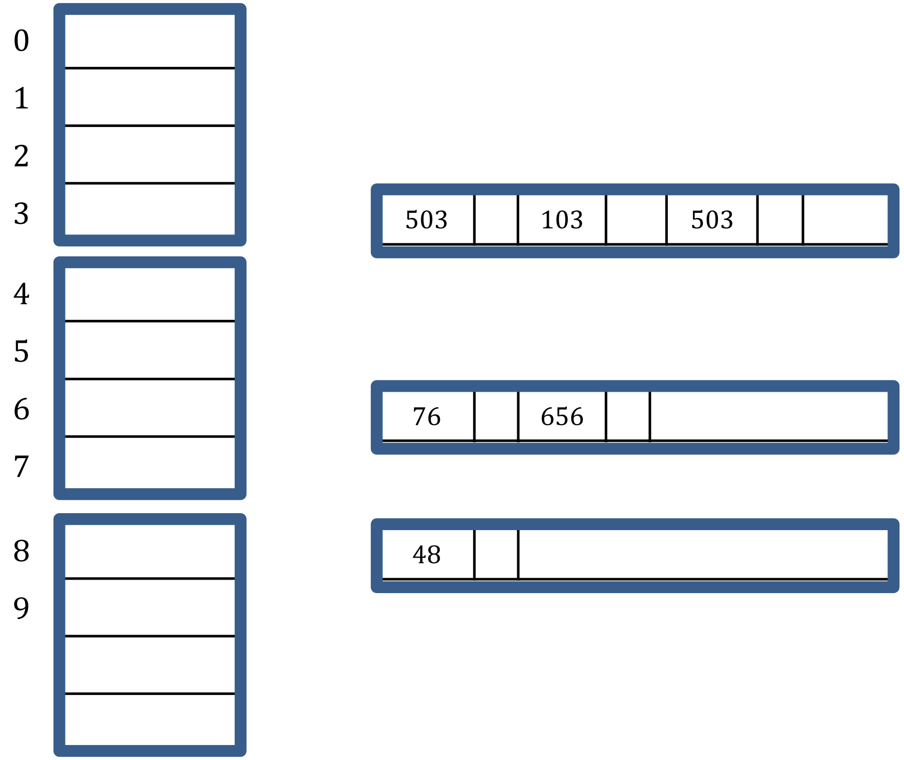


Hash Index

A (naïve) hash function:

$$H(x)=x \bmod 10$$





Hash Index

A (naïve) hash function:

$H(x)=x \bmod 10$

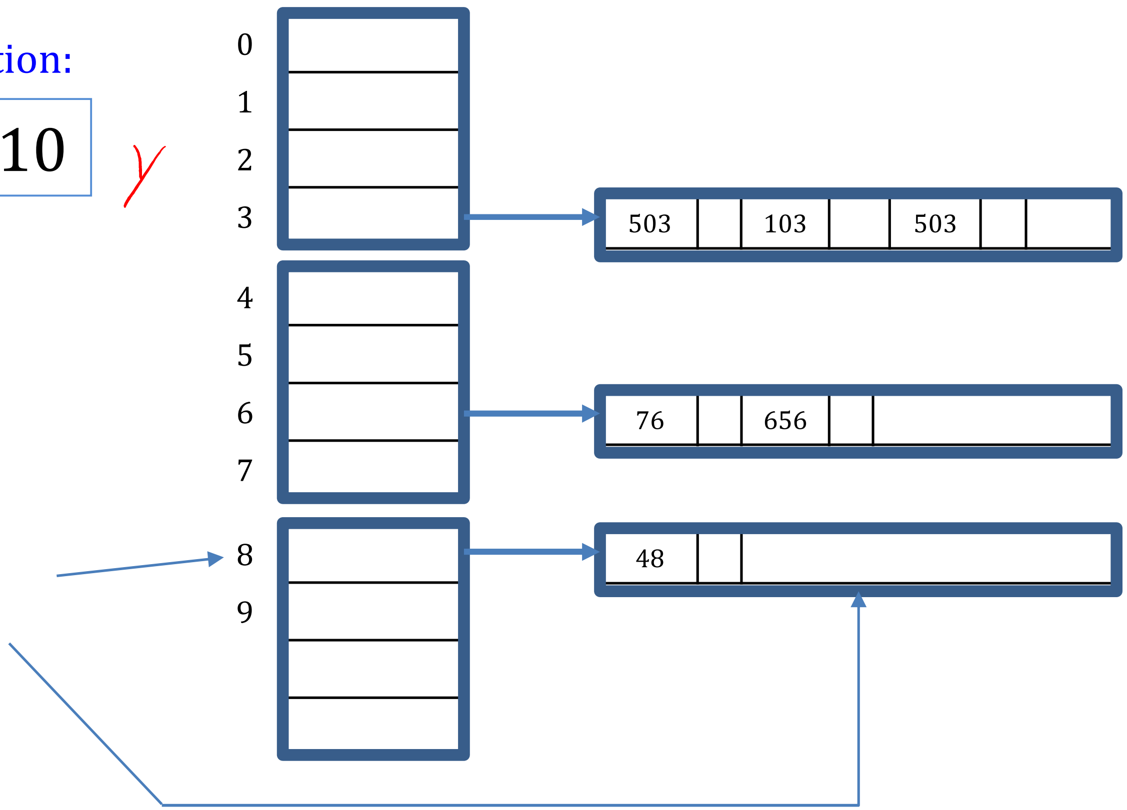
y

= disk block

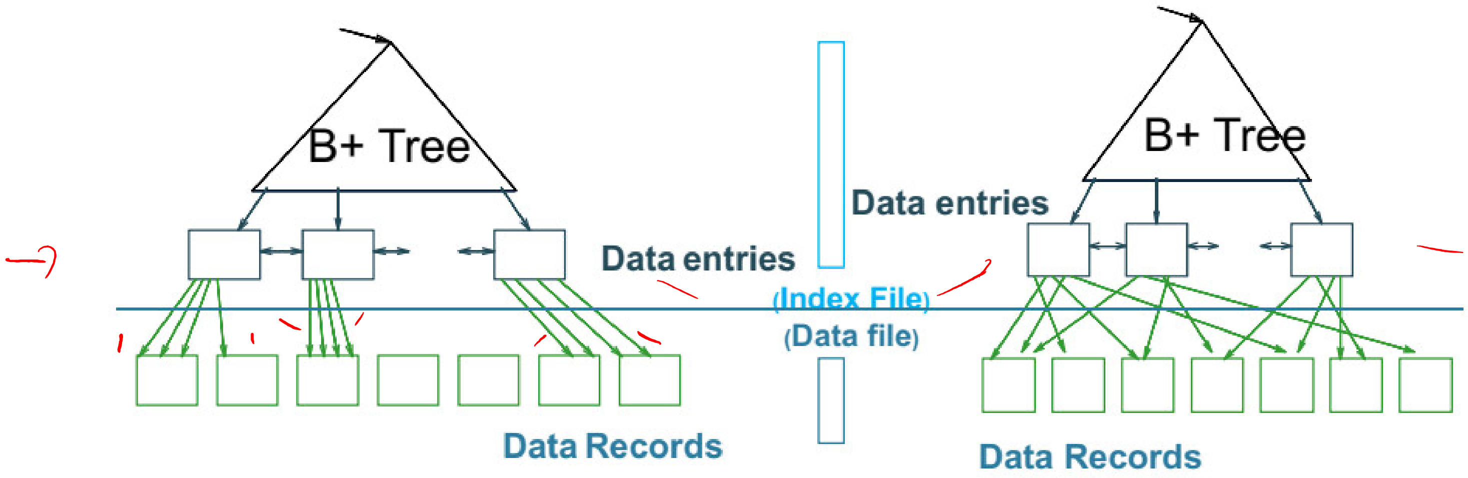
Cost per lookup:

- One access in array
- One access in list

No range queries!



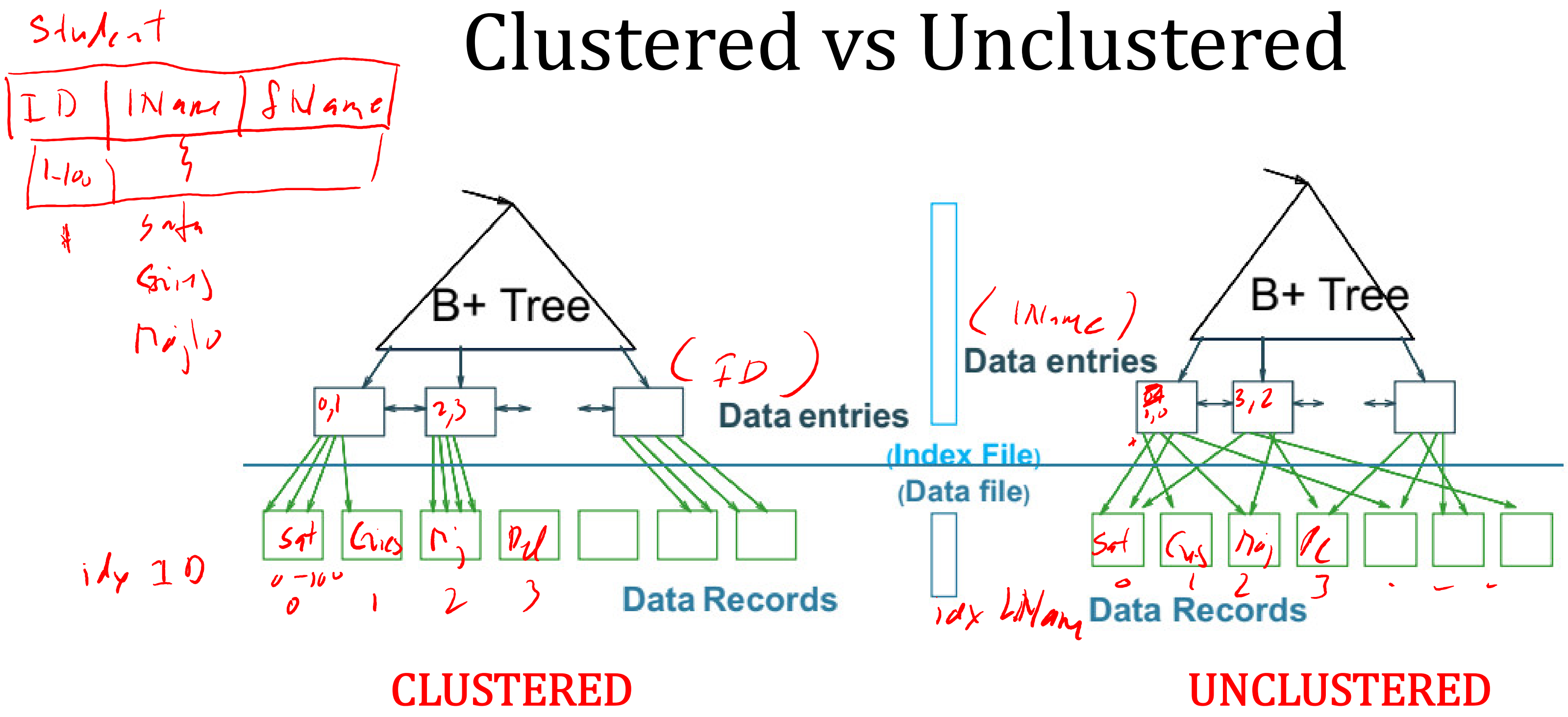
Clustered vs Unclustered



CLUSTERED

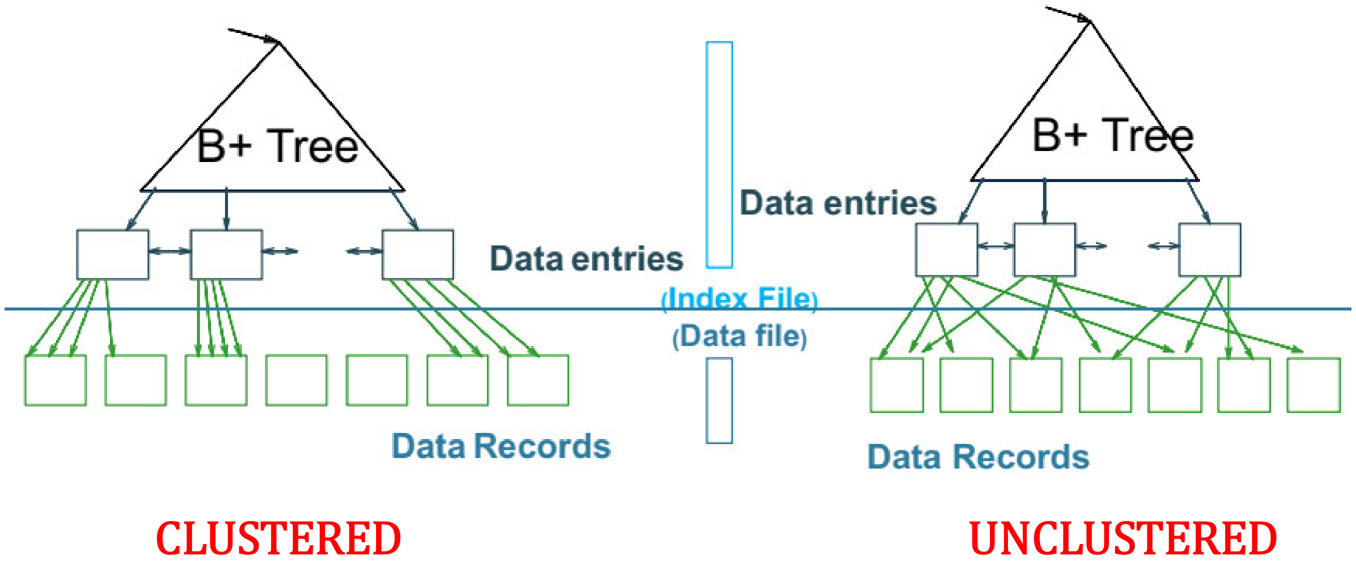
UNCLUSTERED

Clustered vs Unclustered



Every table can have **only one** clustered and **many** unclustered indexes

Clustered vs Unclustered

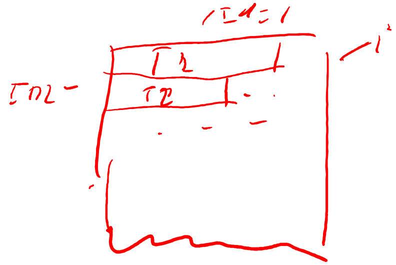


Every table can have **only one** clustered and **many** unclustered indexes

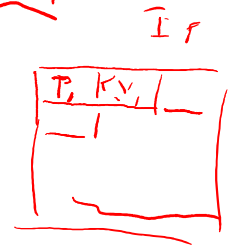
SQL server defaults to cluster by **primary key**

I 1)

Index Classification



- Clustered/unclustered
  - *Clustered records* = records close in index are close in data
    - Option 1: Data inside data file is sorted on disk
    - Option 2: Store data directly inside the index (no separate files)
  - *Unclustered* = records close in index maybe far in data



Index Classification

- Clustered/unclustered
  - *Clustered records* = records close in index are close in data
    - Option 1: Data inside data file is sorted on disk
    - Option 2: Store data directly inside the index (no separate files)
  - *Unclustered* = records close in index maybe far in data
- Primary/ secondary
  - Meaning 1:
    - Primary: is over attributes that include Primary Key
    - Secondary: otherwise
  - Meaning 2: means the same as clustered/unclustered

DB ind

Primary secondary → Physical Logical

Index Classification

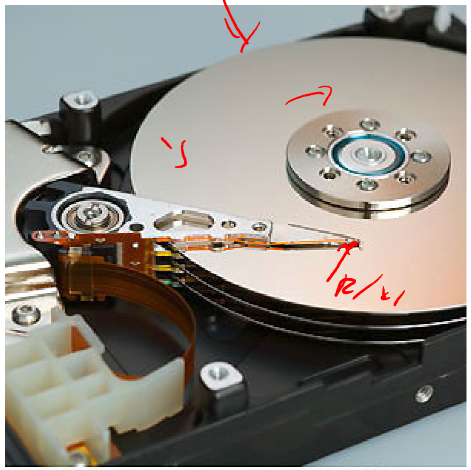
- Clustered/unclustered
  - *Clustered records* = records close in index are close in data
    - Option 1: Data inside data file is sorted on disk
    - Option 2: Store data directly inside the index (no separate files)
  - *Unclustered* = records close in index maybe far in data
- Primary/ secondary
  - Meaning 1:
    - Primary: is over attributes that include Primary Key
    - Secondary: otherwise
  - Meaning 2: means the same as clustered/unclustered
- Organization
  - B+ tree or Hash table

Floppy disk 2.88 MB

Scanning a Data File

TB HD

- Hard disks are mechanical devices!
  - 60's technology, much higher density now
  - Read only as fast as the rotation speed!



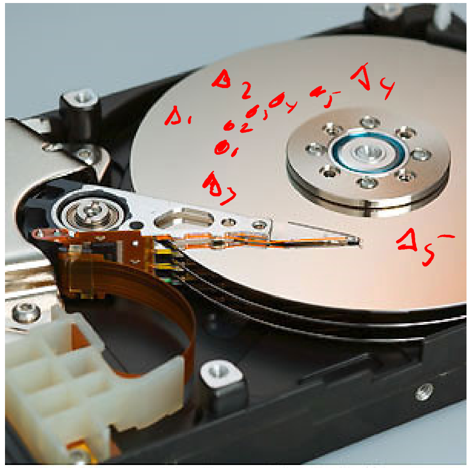
ComputerHope.com

* disk density or areal density - measurement of the amount of data a disk can hold
- measured by the TPI (tracks per inch) or bits per inch

defrag → defragment

Scanning a Data File

- Hard disks are mechanical devices!
  - 60's technology, much higher density now
  - Read only as fast as the rotation speed!
- Result?
  - sequential scan is MUCH FASTER than random reads
  - Good: read blocks 1, 2, 3, 4, 5,...
  - Bad: read blocks 2342, 11, 321, 9, ...

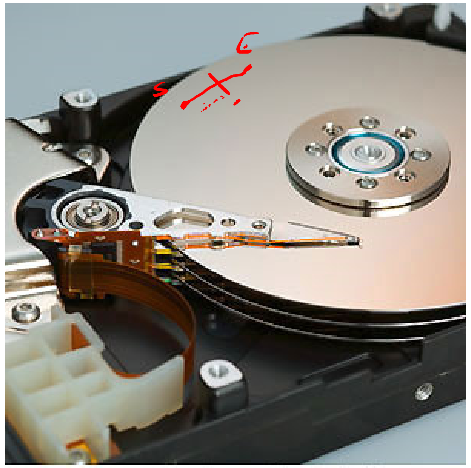


ComputerHope.com

* disk density or areal density - measurement of the amount of data a disk can hold
- measured by the TPI (tracks per inch) or bits per inch

Say, A = 8 FTE2,

Scanning a Data File



ComputerHope.com

- Hard disks are mechanical devices!
  - 60's technology, much higher density now
  - Read only as fast as the rotation speed!
- Result?
  - sequential scan is MUCH FASTER than random reads
  - Good: read blocks 1, 2, 3, 4, 5,...
  - Bad: read blocks 2342, 11, 321, 9, ...
- Rule of thumb:
  - Random reading 1-2% of the file ≈ entire file sequentially scanned
  - this decrease over time (due to increased density of disks)

* disk density or areal density - measurement of the amount of data a disk can hold
- measured by the TPI (tracks per inch) or bits per inch

Scanning a Data File

HDD

- Hard disks are mechanical devices!
  - 60's technology, much higher density now
  - Read only as fast as the rotation speed!
- Result?
  - sequential scan is MUCH FASTER than random reads
  - Good: read blocks 1, 2, 3, 4, 5,...
  - Bad: read blocks 2342, 11, 321, 9, ...
- Rule of thumb:
  - Random reading 1-2% of the file ≈ entire file sequentially scanned
  - this decrease over time (due to increased density of disks)
- HDD ~> Solid State (SSD)
  - Entirely different performance characteristics

ComputerHope.com

* disk density or areal density - measurement of the amount of data a disk can hold
- measured by the TPI (tracks per inch) or bits per inch

Takes(studentID, courseID)
Student(**ID**, name, …)

Example

```
for y in Takes
  if courseID = 200 then
    for x in Student
      if x.ID=y.studentID
        output *
```

```
Takes(studentID, courseID)
Student(ID, name, …)
```

Example

```
for y in Takes
  if courseID = 200 then
    for x in Student
      if x.ID=y.studentID
        output *
```

```
SELECT name
FROM Student x, Takes y
WHERE x.ID=y.studentID
      AND y.courseID=300
```

```
Takes(studentID, courseID)
Student(ID, name, …)
```

Example

```
for y in Takes
  if courseID = 200 then
    for x in Student
      if x.ID=y.studentID
        output *
```

```
SELECT name
FROM Student x, Takes y
WHERE x.ID=y.studentID
      AND y.courseID=300
```

Assume the database has indexes on these attributes:

- **index_takes_course** = index on Takes.courseID
- **index_studentID** = index on Student.ID

```
Takes(studentID, courseID)
Student(ID, name, …)
```

(ID is underlined in Student.)

Example

(Frontend)

Proj

```
for y in Takes
  if courseID = 200 then
    for x in Student
      if x.ID=y.studentID
        output *
```

Sel

(back end)

```
SELECT name
FROM Student x, Takes y
WHERE x.ID=y.studentID
      AND y.courseID=300
```

Assume the database has indexes on these attributes:

- index_takes_course = index on Takes.courseID
- index_studentID = index on Student.ID

Base File

(dBfile idx

key

```
for y1 in index_takes_course where y1.courseID = 300
  for y in y1.Takes
    for x1 in index_studentID where x.ID = y.studentID
      for x in x1.Student
        output x.*, y.*
```

```
Takes(studentID, courseID)
Student(ID, name, …)
```

Example

π

```
for y in Takes
  if courseID = 200 then
    for x in Student
      if x.ID=y.studentID
        output *
```

σ

$\bowtie$

```
SELECT name
FROM Student x, Takes y
WHERE x.ID=y.studentID
      AND y.courseID=300
```

Assume the database has indexes on these attributes:

- **index_takes_course** = index on Takes.courseID
- **index_studentID** = index on Student.ID

Index selection — σ

Index join — $\bowtie_1$

```
for y1 in index_takes_course where y1.courseID = 300
  for y in y1.Takes
    for x1 in index_studentID where x.ID = y.studentID
      for x in x1.Student
        output x.*, y.*
```

π

Getting Practical:
Creating Indexes in SQL

```
CREATE TABLE V(M int, N varchar(20), P int);
```

```
CREATE INDEX V1 ON V(N);
```

```
CREATE INDEX V2 ON V(P, M);
```

```
CREATE INDEX V3 ON V(M, N);
```

```
CREATE UNIQUE INDEX V4 ON V(N);
```

```
CREATE CLUSTERED INDEX V5 ON V(N);
```

Getting Practical:
Creating Indexes in SQL

```
CREATE TABLE V(M int, N varchar(20), P int);
```

```
CREATE INDEX V1 ON V(N);
```

```
CREATE INDEX V2 ON V(P, M);
```

What does this mean?

```
CREATE INDEX V3 ON V(M, N);
```

```
CREATE UNIQUE INDEX V4 ON V(N);
```

```
CREATE CLUSTERED INDEX V5 ON V(N);
```

Not supported in SQLite

Student

Id	fName	lName
10	Divan	Mahlooji
20	Jenny	Rhee
…	…	…

Which Indexes?

- How many indexes **could** we create?

Student

Id	fName	lName
10	Divan	Mahlooji
20	Jenny	Rhee
…	…	…

Which Indexes?

- How many indexes **could** we create?

15, namely: (ID), (fName), (lName), (ID, fName), (fName, ID), …

(ID, lM) (LM, ID) (f, LN) (LN, FN), ~~(FD, LN, FN)~~

Student

Id	fName	lName
10	Divan	Mahlooji
20	Jenny	Rhee
…	…	…

Which Indexes?

- How many indexes **could** we create?

15, namely: (ID), (fName), (lName), (ID, fName), (fName, ID), …

- Which indexes **should** we create?

5? 10? 15; 20?

Student

Id	fName	lName
10	Divan	Mahlooji
20	Jenny	Rhee
…	…	…

Which Indexes?

- How many indexes **could** we create?

15, namely: (ID), (fName), (lName), (ID, fName), (fName, ID), …

- Which indexes **should** we create?

Few! Each new index slows down updates to Student

Index selection is a hard problem

Student

Id	fName	lName
10	Divan	Mahlooji
20	Jenny	Rhee
…	…	…

Which Indexes?

- The index selection problem
  - given a Studn table, and a "workload" (big Java application with lots of SQL queries),
  - decide which indexes to create (and which ones NOT to create!)

Student

Id	fName	lName
10	Divan	Mahlooji
20	Jenny	Rhee
…	…	…

Which Indexes?

- The index selection problem
  - given a table, and a "workload" (big Java application with lots of SQL queries),
  - decide which indexes to create (and which ones NOT to create!)

- Who will do index selection?
  - database administrator DBA
  - semi-automatically, using a database administration tool

Index Selection: Which Search Key?

- Make some attribute **K** a search key if the WHERE clause contains:
  - an exact match on **K**   K = '20200512'
  - a range predicate on **K**   K = '20200512 — 20200612'
  - a join on **K**

Index Selection Problem

```
V(M, N, P);
```

```
SELECT *
FROM V
WHERE V.M = 33;
```

Scan V
For each record:
  if M=33 **then** output

Lookup key 33 in I1
For each record: output

INDEX I1 **ON** V(M)

```
SELECT *
FROM V
WHERE V.M = 33
      AND V.P = 55;
```

Scan V
For each record:
  if M=33 and P=55 **then** output

Lookup key 33 in I1
For each record
  if P=55 **then** output

Index Selection Problem 1

```
V(M, N, P);
```

The workload is just as given and nothing else:

100,000 queries:

```
SELECT *
FROM V
WHERE N = ?
```

10K

100 queries:

```
SELECT *
FROM V
WHERE P = ?
```

Which indexes?

Index Selection Problem 1

```
V(M, N, P);
```

The workload is just as given and nothing else:

100,000 queries:

```sql
SELECT *
FROM V
WHERE N = ?
```

100 queries:

```sql
SELECT *
FROM V
WHERE P = ?
```

A: **V(N)** and **V(P)** (hash tables or B-trees)

Index Selection Problem 2

```
V(M, N, P);
```

The workload is just as given and nothing else:

100,000 queries:

```
SELECT *
FROM V
WHERE N > ?
  AND N < ?
```

100 queries:

```
SELECT *
FROM V
WHERE P = ?
```

100,000 queries:

```
INSERT INTO V
VALUES(?, ?, ?)
```

Which indexes?

Index Selection Problem 2

```
V(M, N, P);
```

The workload is just as given and nothing else:

100,000 queries:

```
SELECT *
FROM V
WHERE N > ?
  AND N < ?
```

v(N)

100 queries:

```
SELECT *
FROM V
WHERE P = ?
```

v(P)

100,000 queries:

```
INSERT INTO V
VALUES(?, ?, ?)
```

↳ v(M), v(N), v(P)

A: definitely **V(N)** (must B-tree); unsure about **V(P)**

Index Selection Problem 3

```
V(M, N, P);
```

The workload is just as given and nothing else:

100,000 queries:

```
SELECT *
FROM V
WHERE N = ?
```

V(N)

1,000,000 queries:

```
SELECT *
FROM V
WHERE N = ?
  AND P > ?
```

V(N), V(P) ⇒ V(N,P)

100,000 queries:

```
INSERT INTO V
VALUES(?, ?, ?)
```

V(N,P) = 8

n = ?

P = ?

Which indexes?

V(N), V(N,P) ⇒ V(N,P)

Index Selection Problem 3

```
V(M, N, P);
```

The workload is just as given and nothing else:

100,000 queries:

```
SELECT *
FROM V
WHERE N = ?
```

1,000,000 queries:

```
SELECT *
FROM V
WHERE N = ?
  AND P > ?
```

100,000 queries:

```
INSERT INTO V
VALUES(?, ?, ?)
```

A: **V(N, P)** (B-tree)

Index Selection Problem 3

```
V(M, N, P);
```

The workload is just as given and nothing else:

100,000 queries:

```
SELECT *
FROM V
WHERE N = ?
```

1,000,000 queries:

```
SELECT *
FROM V
WHERE N = ?
  AND P > ?
```

H.W.

100,000 queries:

```
INSERT INTO V
VALUES(?, ?, ?)
```

A: **V(N, P)** (B-tree)

How does this index differ from:

1. Two indexes **V(N)** and **V(P)**?
2. An index **V(P, N)**?

Index Selection Problem 4

```
V(M, N, P);
```

The workload is just as given and nothing else:

~~1,000~~ 100 queries:

```
SELECT *
FROM V
WHERE N > ?
  AND N < ?
```

100,000 queries:

```
SELECT *
FROM V
WHERE P > ?
  AND P < ?
```

V(N) → S.I.

Which indexes?

V(P) → P I

Index Selection Problem 4

```
V(M, N, P);
```

The workload is just as given and nothing else:

1,000 queries:

```
SELECT *
FROM V
WHERE N > ?
  AND N < ?
```

100,000 queries:

```
SELECT *
FROM V
WHERE P > ?
  AND P < ?
```

A: **V(N)** secondary, **V(P)** primary index (both B-tree)

Index Selection Problem 5

```
V(M, N, P);
```

Q1 ✓

```
SELECT *
FROM V
WHERE V.M = 33
```

Q2 ✓

```
SELECT *
FROM V
WHERE V.M = 33
  AND V.P = 55
```

Suppose the database has these indexes. Which ones can the optimizer use?

Yes

Yes

```
INDEX I1 ON V(M);
INDEX I2 ON V(M, P);
INDEX I3 ON V(P, M);
```

A

B

C

Index Selection Problem 5 – Recap Indexes

```
V(M, N, P);
```

Suppose the database has these indexes. Which ones can the optimizer use?

Q1

```
SELECT *
FROM V
WHERE V.M = 33
```

Q2

```
SELECT *
FROM V
WHERE V.M = 33
  AND V.P = 55
```

Yes!

```
INDEX I1 ON V(M);

INDEX I2 ON V(M, P);

INDEX I3 ON V(P, M);
```

	Q1	Q2
I1	✓	✓
I2		

Index Selection Problem 5 – Recap Indexes

```
V(M, N, P);
```

I_1 P_I

Q1

```
SELECT *
FROM V
WHERE V.M = 33
```

Q2

```
SELECT *
FROM V
WHERE V.M = 33
  AND V.P = 55
```

Yes! But why?

Yes!

Suppose the database has these indexes. Which ones can the optimizer use?

```
INDEX I1 ON V(M);

INDEX I2 ON V(M, P);

INDEX I3 ON V(P, M);
```

P

P S

Q1 Q2

I3

Index Selection Problem 5 – Recap Indexes

```
V(M, N, P);
```

```
SELECT * 
FROM V
WHERE V.M = 33
```

```
SELECT *
FROM V
WHERE V.M = 33
  AND V.P = 55
```

Suppose the database has these indexes. Which ones can the optimizer use?

```
INDEX I1 ON V(M);

INDEX I2 ON V(M, P);

INDEX I3 ON V(P, M);
```

No! But why?

Yes!

Recap Indexes

```
Movie(mid, title, year)
```

CLUSTERED INDEX I ON Movie(id)
INDEX J ON Movie(year)

```
SELECT *
FROM Movie
WHERE year = 2010
```

'J' ✓

```
SELECT *
FROM Movie
WHERE (year = 1910)
```

'J' X

title = 'RATATOUILLE';

The system used the index **J** for one of the queries, but not for the other.

Which and why?

Basic Index Selection Guidelines

- Consider queries in workload in order of importance
  - ignore infrequent queries if you also have many writes

Basic Index Selection Guidelines

- Consider queries in workload in order of importance
  - ignore infrequent queries if you also have many writes
- Consider relations accessed by query
  - No point indexing other relations

Basic Index Selection Guidelines

- Consider queries in workload in order of importance
  - ignore infrequent queries if you also have many writes
- Consider relations accessed by query
  - No point indexing other relations
- Look at WHERE clause for possible search key

S ~
F ~
W year = '2010'

S * ST
F
W SID = "xxx"

Basic Index Selection Guidelines

- Consider queries in workload in order of importance
  - ignore infrequent queries if you also have many writes
- Consider relations accessed by query
  - No point indexing other relations
- Look at WHERE clause for possible search key
- Try to choose indexes that speed-up multiple queries

To Cluster or Not to Cluster

- Range queries benefit mostly from clustering

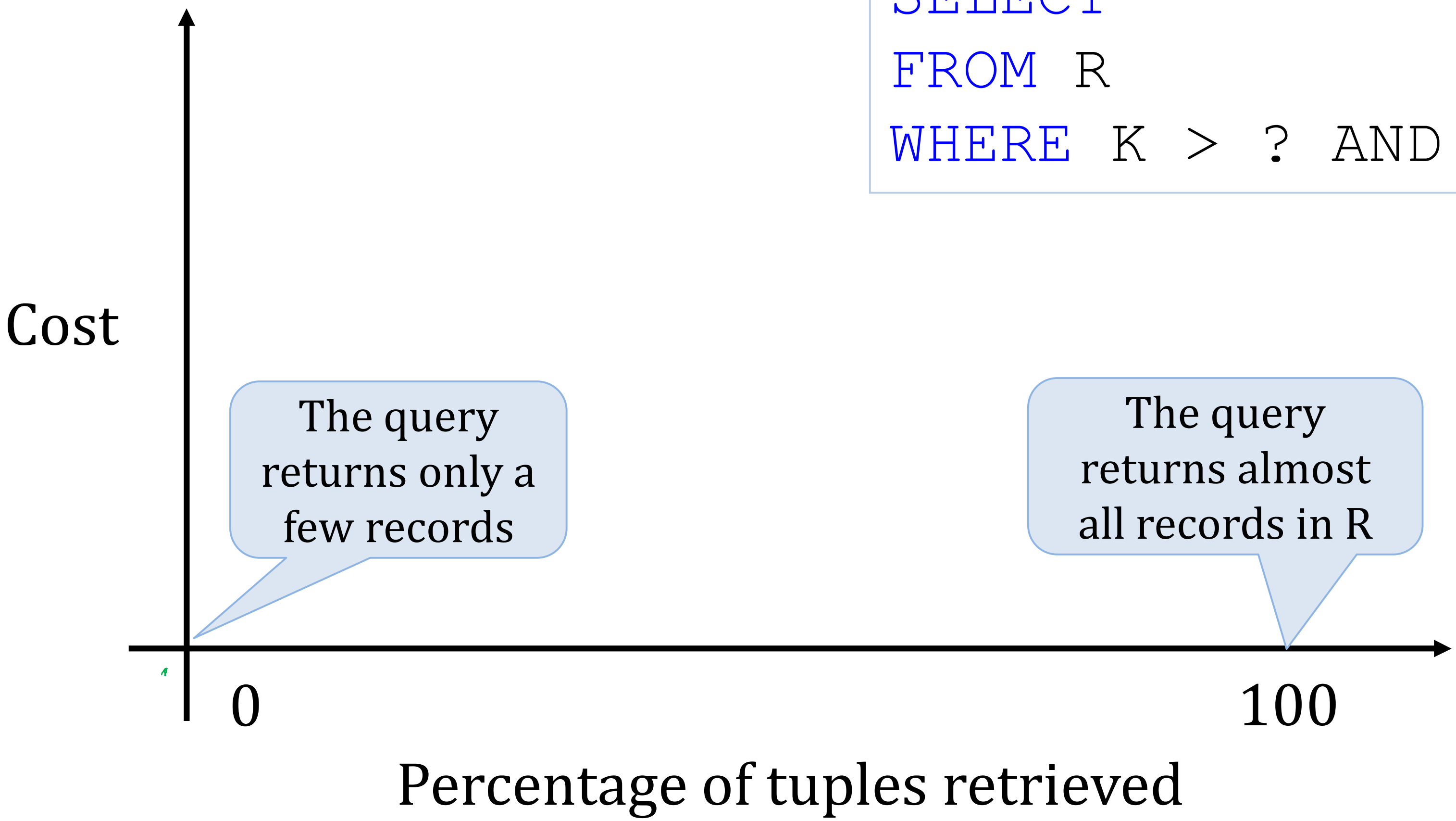
To Cluster or Not to Cluster

- Range queries benefit mostly from clustering

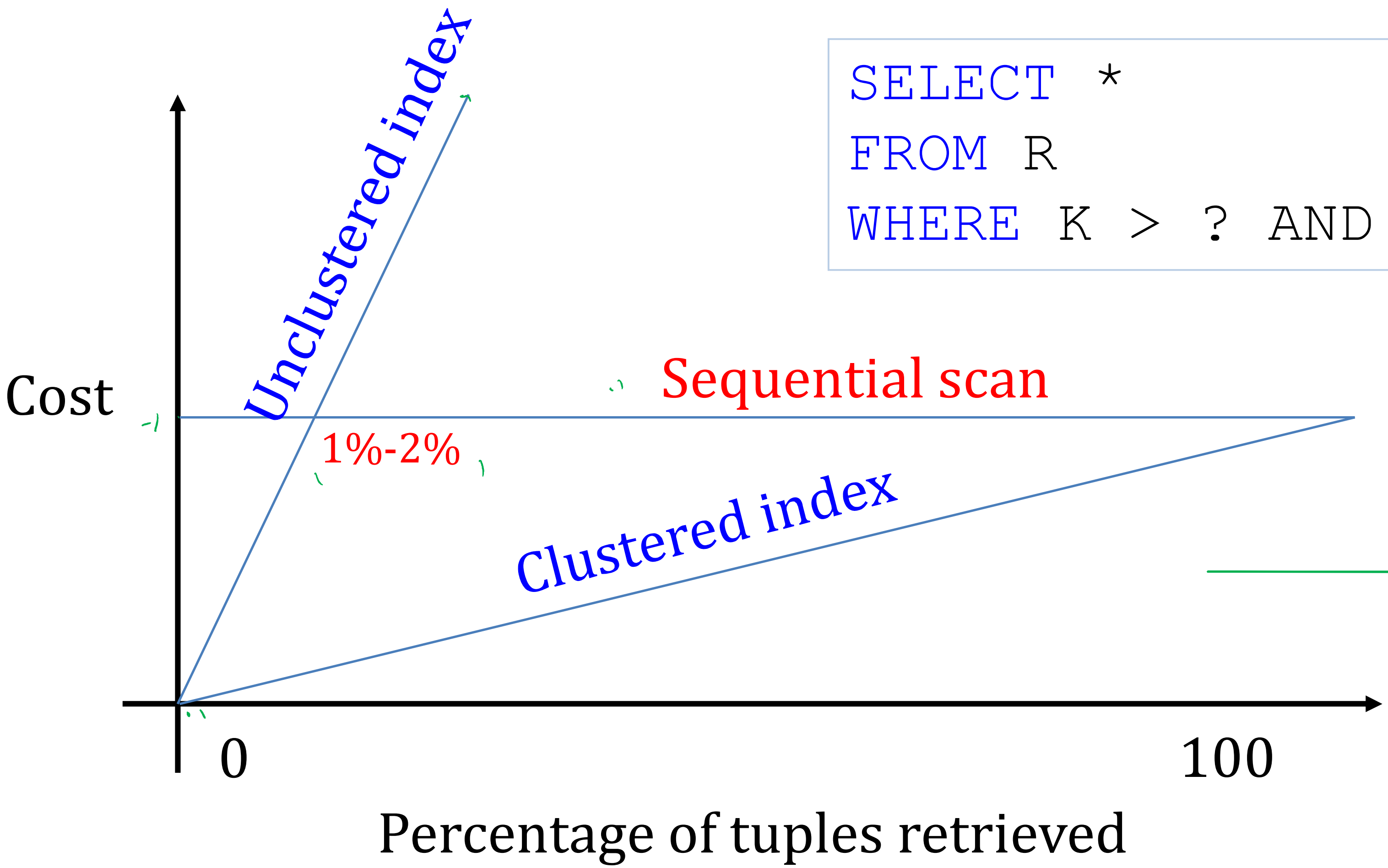
- Covering indexes do not need to be clustered: they work equally well unclustered
  - a covering index for a query is one where every attribute mentioned in the query is part of the index's search key
  - in that case, index has all the info you need anyway

To Cluster or Not to Cluster

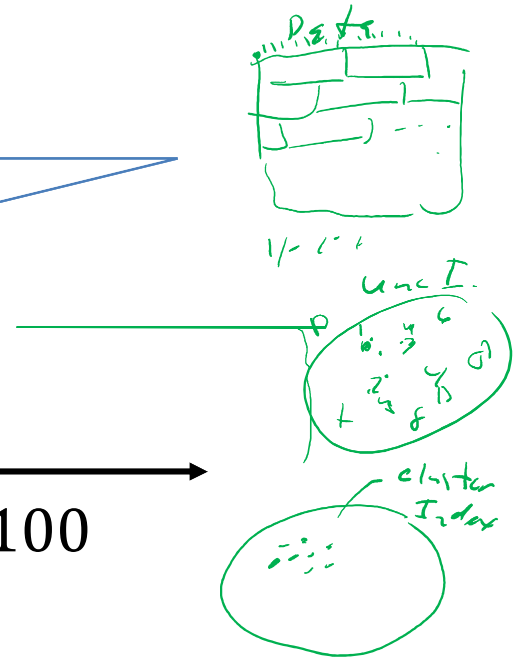
```
SELECT *
FROM R
WHERE K > ? AND K < ?
```



To Cluster or Not to Cluster




```
SELECT *
FROM R
WHERE K > ? AND K < ?
```



Thank you.