Mod 5 - Lecture 7,8,9:

Secondary Indexes Multi-Level Indexes

- Deepak Poonia
(IISc Bangalore; GATE AIR 53 & 67)

Content of this Lecture:

1. Secondary Indexes
2. Multilevel Indices
3. ISAM

DBMS Complete Course Link:

https://www.goclasses.in/courses/Database-Management-Systems

Instructor:

Deepak Poonia

IISc Bangalore

GATE CSE AIR 53; AIR 67; AIR 206; AIR 256

DBMS Complete Course Link:

https://www.goclasses.in/courses/Database-Management-Systems

GO CLASSES

GATE 2024
COMPLETE COURSE CS-IT
(1 YEAR + 3 MONTHS)

GATE 2024 COMPLETE COURSE CS - IT

(1 YEAR + 3 MONTHS)

GO CLASSES

GATE 2023
COMPLETE COURSE CS-IT
(6 MONTHS)

GATE 2023 COMPLETE COURSE CS-IT

(6 MONTHS)

GO CLASSES

GATE 2025
COMPLETE COURSE CS-IT
(2 YEARS)

GATE 2025 COMPLETE COURSE CS - IT

(2 YEARS + 3 MONTHS)

2023 Discrete Mathematics

★ ★ ★ ★ ★ 5.0 (62 ratings)

Deepak Poonia (MTech IISc Bangal...

Free

2023 C Programming

★ ★ ★ ★ ★ 5.0 (59 ratings)

Sachin Mittal (MTech IISc Bangalor...

Free

2023 Discrete Mathematics

Learn, Understand, Discuss. "GO" for the Best.

★★★★★ 5.0 (62 ratings)

5997 Learners Enrolled

Language: English

Instructors: Deepak Poonia (MTech IISc Bangalore, GATE CSE AIR 53; 67)

FREE

On “GATE-Overflow ” Website

Join **GO+ GO Classes Combined Test Series** for BEST quality tests, matching GATE CSE Level:

Visit www.gateoverflow.in website to join Test Series.

1. **Quality Questions:** No Ambiguity in Questions, All Well-framed questions.
2. Correct, **Detailed Explanation**, Covering Variations of questions.
3. **Video Solutions.**

NOTE :

Complete Discrete Mathematics & Complete C-Programming

Courses, by GO Classes, are **FREE** for ALL learners.

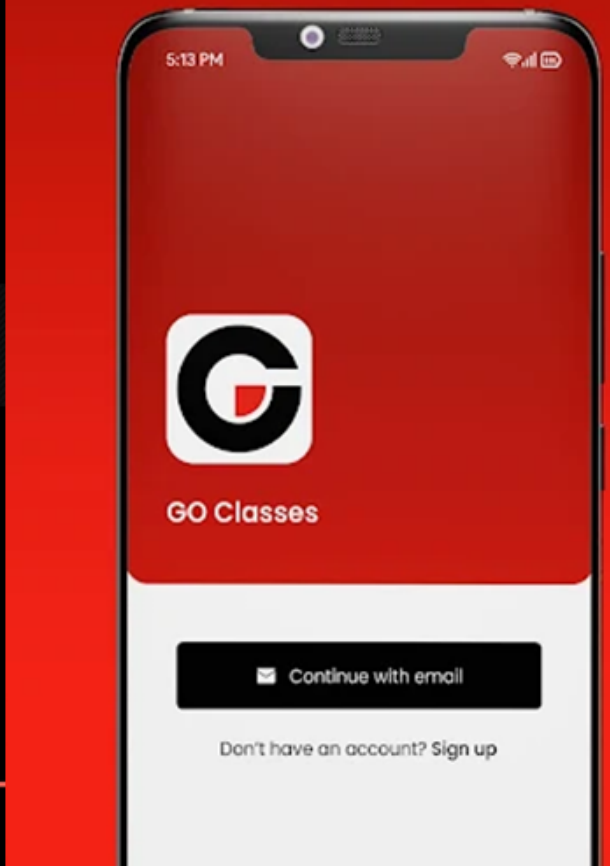
Visit here to watch : https://www.goclasses.in/s/store/

SignUp/Login on Goclasses website for free and start learning.

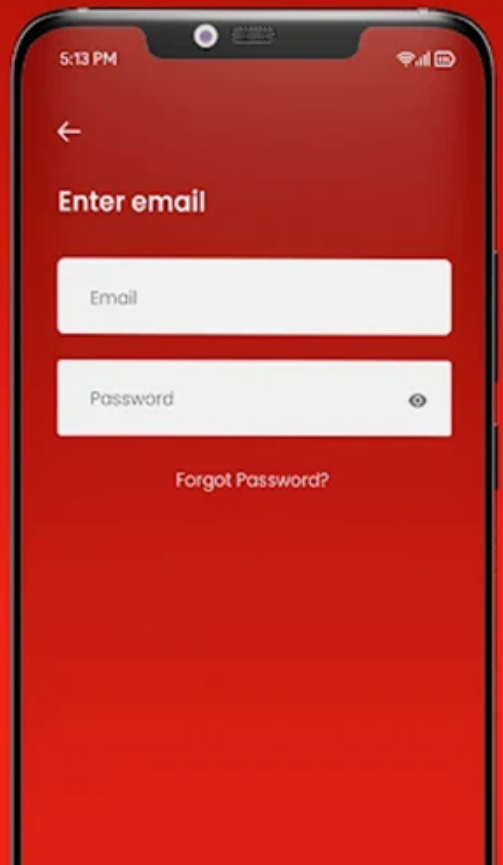
Download the GO Classes Android App:

https://play.google.com/store/apps/details?id=com.goclasses.courses

Search "GO Classes" on Play Store.

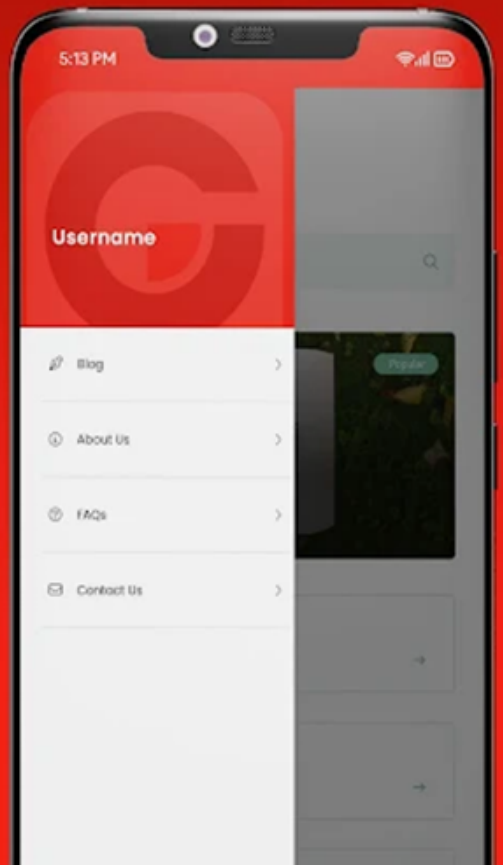


Welcome to
GO Classes
Learn at your own pace



Learn from
Top educators
Anytime, anywhere!

Easy to use
Navigation Panel
with all essentials



We are on Telegram. Contact Us for any help.

Join GO Classes Resources, Notes, Content, information Telegram Channel:

Public Username: GOCLASSES_CSE

Join GO Classes **Doubt Discussion** Telegram Group :

Username: GATECSE_Goclasses

(Any doubt related to Goclasses Courses can also be asked here.)

Join GATEOverflow Doubt Discussion Telegram Group :

Username: gateoverflow_cse

Types of Single level Ordered Indexes:

2. Clustering Index

Clustering Index:

If the file containing the records is sequentially ordered by a Non-key field F, then the Index using search key F is called Clustring Index.

17.1.2 Clustering Indexes

If file records are physically ordered on a nonkey field—which *does not* have a distinct value for each record—that field is called the **clustering field** and the data file is called a **clustered file.** We can create a different type of index, called a **clustering index**, to speed up retrieval of all the records that have the same value for the clustering field. This differs from a primary index, which requires that the ordering field of the data file have a *distinct value* for each record.

Q:

Consider a Heap file (Do you remember what a Heap file is ???)

Can we create a Clustering Index on a heap file ??

Ans: NO. File must be Physically Ordered by a Non-key attribute.

NOTE:

To create Clustering Index on a file R:

R MUST be PHYSICALLY Ordered by a Non-Key Field F...

Then a Index by F is called Clustering Index.

Q:

How many Clustering Indexes can be created on a file ?

Ans: At Most One

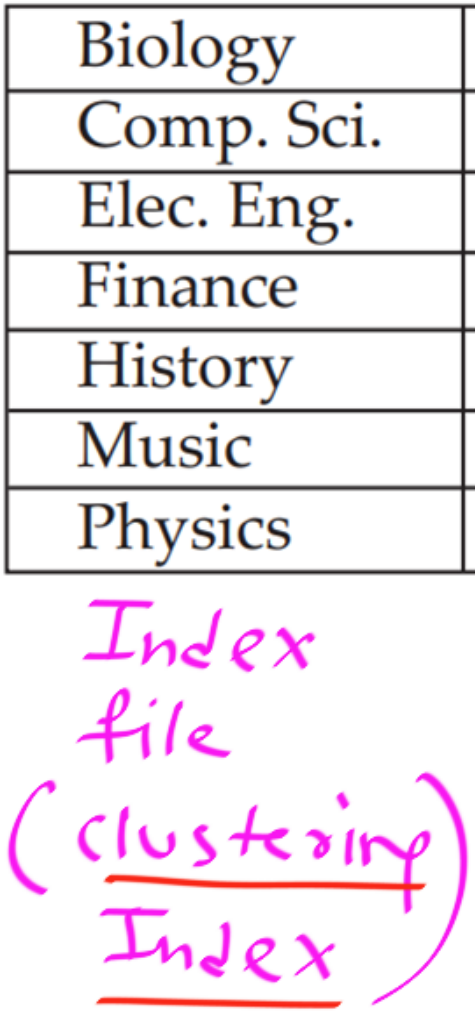
Q:

On a file, Can we create both Primary as well as Clustering Index ??

Ans: NO. At most one of them.

Clustering Index:

Implementation



76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
45565	Katz	Comp. Sci.	75000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000
12121	Wu	Finance	90000
76543	Singh	Finance	80000
32343	El Said	History	60000
58583	Califieri	History	62000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
33465	Gold	Physics	87000

Clustering Index:

< Search key value , Block pointer >

↙ for every Distinct Search key value we make one Index entry.

↘ Block address of that Block in which first occurrence of Search key value exists.

#Index Entries(Records) =

#clusters in Data file = #Distinct values of ONKF

{ Ordering Non-key field

Clustering Index

Built on ordered files where ordering field is *not a key*

Index attribute: ordering field (OF)

Index entry:

Distinct value V_i of the OF	address of the first block that has a record with OF value V_i

Index file: Ordered file (sorted on OF)

size – no. of distinct values of OF

A clustering index is also an ordered file with two fields; the first field is of the same type as the clustering field of the data file, and the second field is a disk block pointer. There is one entry in the clustering index for each *distinct value* of the clustering field, and it contains the value and a pointer to the *first block* in the data file that has a record with that value for its clustering field. Figure 17.2 shows an

A clustering index is another example of a *nondense* index because it has an entry for every *distinct value* of the indexing field, which is a nonkey by definition and hence has duplicate values rather than a unique value for every record in the file.

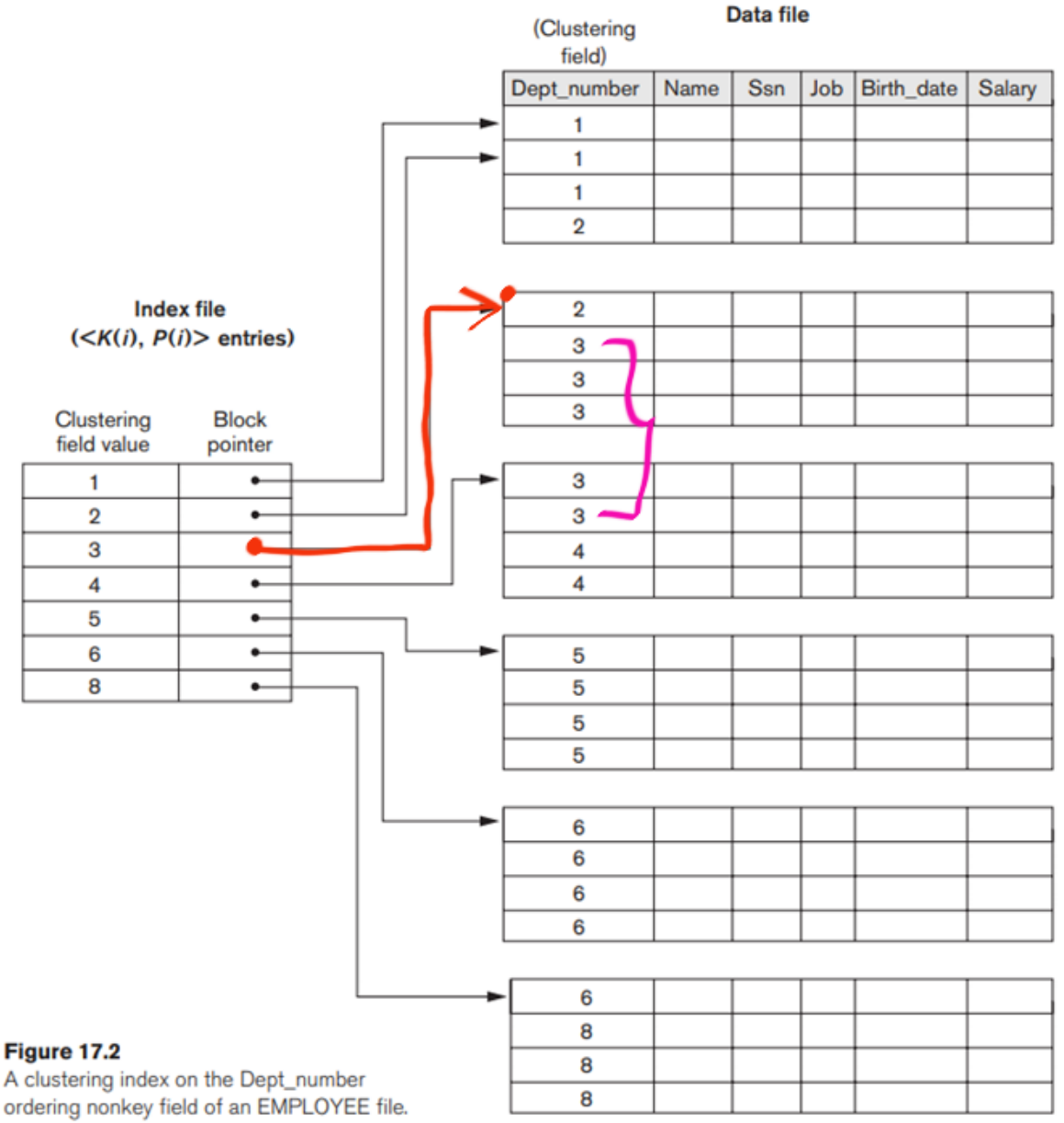
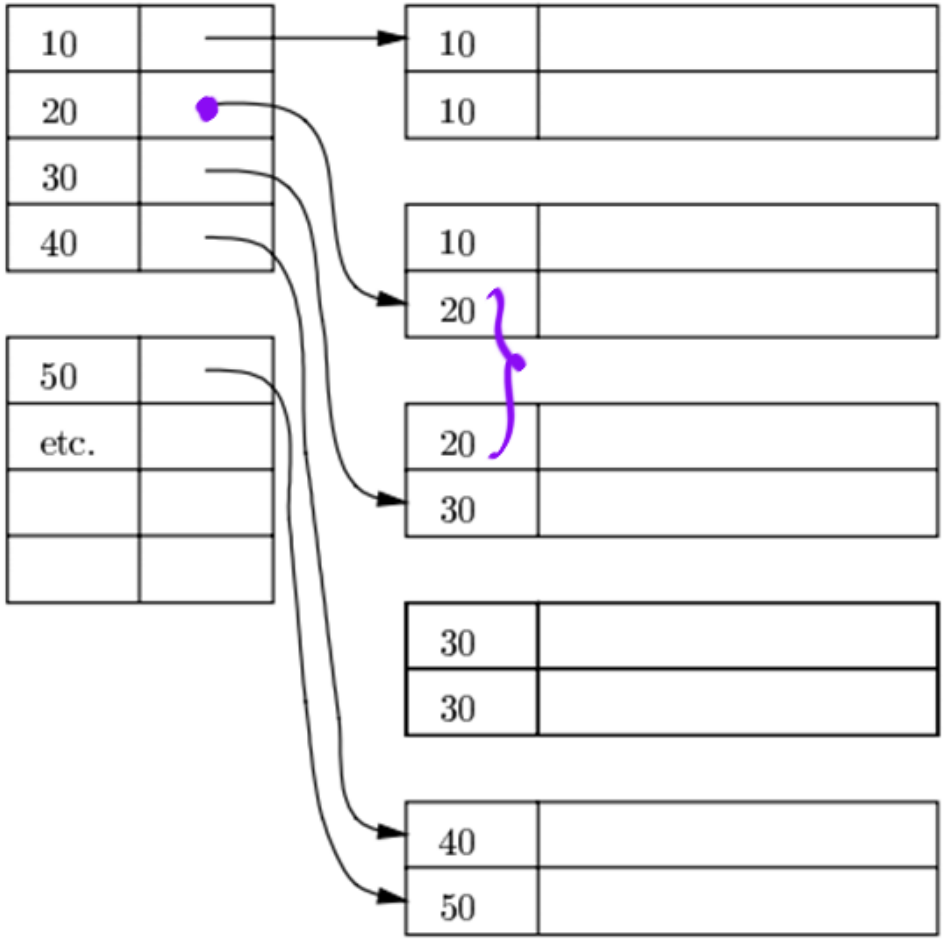
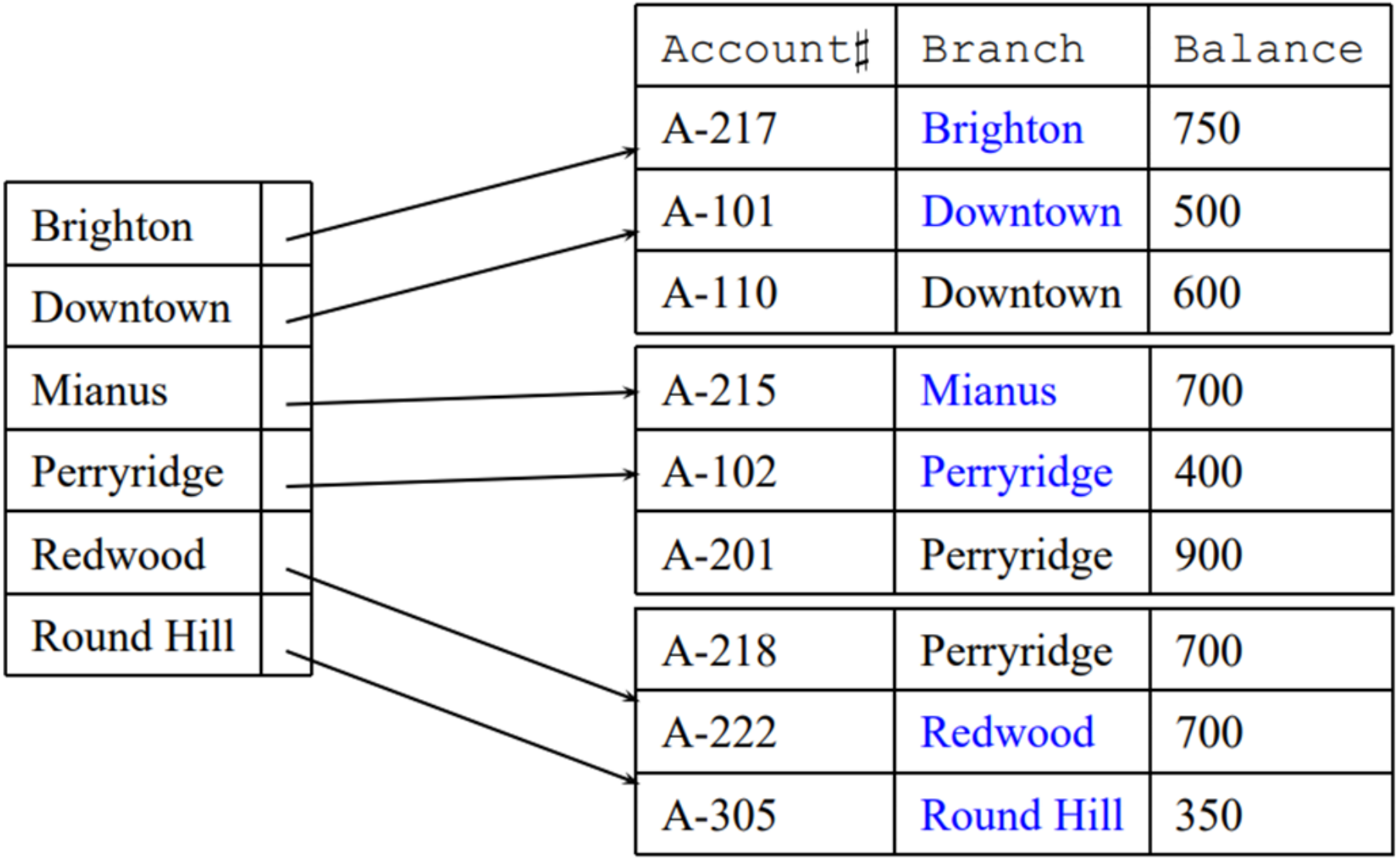


Figure 17.2
A clustering index on the Dept_number ordering nonkey field of an EMPLOYEE file.

Lookup. Find the search key on the index, read the pointed disk block, possibly read successive blocks.




Brighton
Downtown
Mianus
Perryridge
Redwood
Round Hill
Account#
Branch
Balance
A-217
Brighton
750
A-101
Downtown
500
A-110
Downtown
600
A-215
Mianus
700
A-102
Perryridge
400
A-201
Perryridge
900
A-218
Perryridge
700
A-222
Redwood
700
A-305
Round Hill
350

NOTE:

Clustering Index is a Sparse Index.

In Clustering index, we make One index entry for Every Distinct Value of search key in the data file.

Q: Suppose that we consider a ordered file with r = 300,000 records stored on a disk with block size B = 4,096 bytes. Imagine that it is ordered by the attribute Zipcode and there are 1,000 zip codes in the file (with an average 300 records per zip code, assuming even distribution across zip codes.) Assume 5-byte Zipcode and 6-byte block pointer. Data Record Size = 100 B

We create Index on Zipcode. Number of Index entries? Blocking Factor of the Index?? Number of Index Blocks?

Data file : Ordered by Zipcode

Zipcode → Non-key field

Index on Zipcode : Clustering Index

① # Index Entries = # Distinct Zipcode values

= 1000

(2) Index Entry (Record) size =

(search key) Zipcode size + Block address size

= 5 + 6 = 11 B

(3) Blocking factor of Index :

Block Size = 4096 B

Index entries per Block = $\left\lfloor \frac{4096}{11} \right\rfloor$ = 372

(4) # Index Blocks Required:

Index: 1000 Records

Per Block ⇒ 372 Index Records

$$\#\text{Index Blocks} = \left\lceil \frac{1000}{372} \right\rceil = 3$$

5 Searching ZipCode = 302017

↓ #Disk Block Access ? (without Index)

max

(5) Searching ZipCode = 302017

↓ #Disk Block Access ? (without Index)

max

→ #Data Blocks

$\lceil \log_2 B \rceil$

file is sorted by ZipCode.

& ↓ Searching on zipcode you Binary Search

7500

= (13 Block Access) + ??

Data file :

Record Size = 100 B

Block " = 4096 B

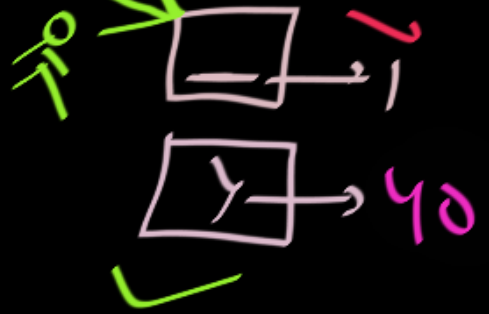
#Records per Block = Block factor of file $= \left\lfloor \frac{4096}{100} \right\rfloor = 40$

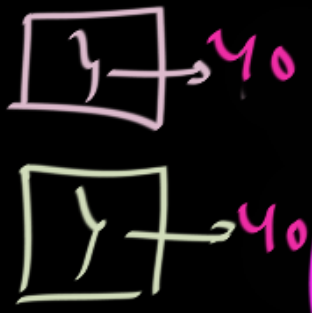
#Blocks for file = $\left\lceil \frac{300,000}{40} \right\rceil = 7500$ file Blocks

Zipcode = 302017 Records we want to find

13 Block Access + 8 = 21 Block Access (max)

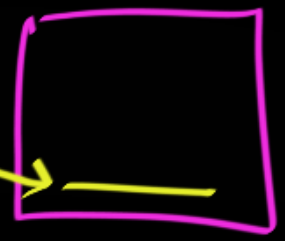
(first occurrence of this zipcode)





$\lceil \frac{299}{40} \rceil = 8$

First Record



299 Records → 40 Per Block

$\lceil \frac{299}{40} \rceil = 8$ more Blocks

(5) Searching ZipCode = 302017

with Index:

Clustering

$\lceil \log_2 I \rceil$ + 9 Data Blocks

3

Index Block Access = 2

Max Block Access

11

(6) If Index is Already in main memory then 9 Disk Block Access (max.)

Example 2. Suppose that we consider the same ordered file with $r = 300{,}000$ records stored on a disk with block size $B = 4{,}096$ bytes. Imagine that it is ordered by the attribute Zipcode and there are 1,000 zip codes in the file (with an average 300 records per zip code, assuming even distribution across zip codes.) The index in this case has 1,000 index entries of 11 bytes each (5-byte Zipcode and 6-byte block pointer) with a blocking factor $bfr_i = \lfloor (B/R_i) \rfloor = \lfloor (4{,}096/11) \rfloor = 372$ index entries per block. The number of index blocks is hence $b_i = \lceil (r_i/bfr_i) \rceil = \lceil (1{,}000/372) \rceil = 3$ blocks. To perform a binary search on the index file would need $\lceil (\log_2 b_i) \rceil = \lceil (\log_2 3) \rceil = 2$ block accesses. Again, this index would typically be loaded in main memory (occupies 11,000 or 11 Kbytes) and takes negligible time to search in memory. One block access to the data file would lead to the first record with a given zip code.

Types of Single level Ordered Indexes:

3. Secondary Index

File R :

→ Physically Stored on Disk

→ field B is Unordered in Data file R

Index on B

→ Secondary Index.

Secondary Index :

Ordered

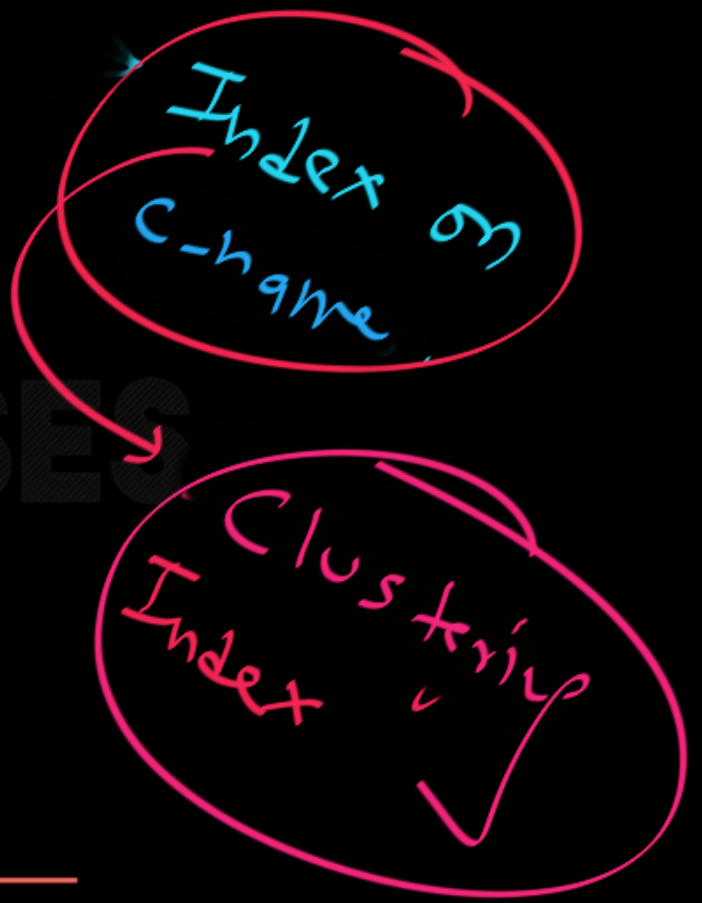
Index on Non-ordering field of Data file

1. If B key : SI on key
2. If B Nonkey : SI on Non-key

Assume that it is also physical order :

customer_name	account_number
Hayes	A-102
Johnson	A-101
Johnson	A-201
Jones	A-217
Lindsay	A-222
Smith	A-215
Turner	A-305

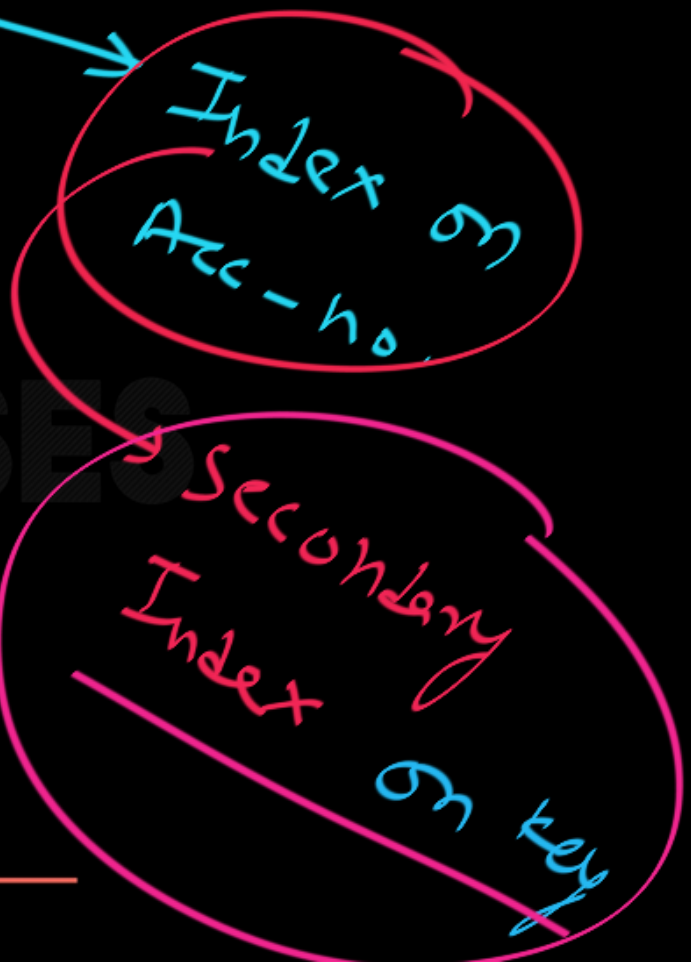
physical order of Data file



Assume that it is also physical order :

customer_name	account_number
Hayes	A-102
Johnson	A-101
Johnson	A-201
Jones	A-217
Lindsay	A-222
Smith	A-215
Turner	A-305

physical order of Data file



Assume that it is also physical order :

	customer_name	account_number
B_1	Hayes	A-102
B_1	Johnson	A-101
B_2	Johnson	A-201
B_2	Jones	A-217
B_3	Lindsay	A-222
B_3	Smith	A-215
B_4	Turner	A-305

Physical order of Data file

B_1 B_1 B_2 B_3 B_4 Block addresses

Secondary Index on Acc-no (Unordered)

A-101	B_1
A-102	B_1
A-201	B_2
A-215	B_3
A-217	B_2
A-222	B_3
A-305	B_4

Assume that it is also physical order :

	customer_name	account_number
R_1	Hayes	A-102
R_2	Johnson	A-101
R_3	Johnson	A-201
R_4	Jones	A-217
R_5	Lindsay	A-222
R_6	Smith	A-215
R_7	Turner	A-305

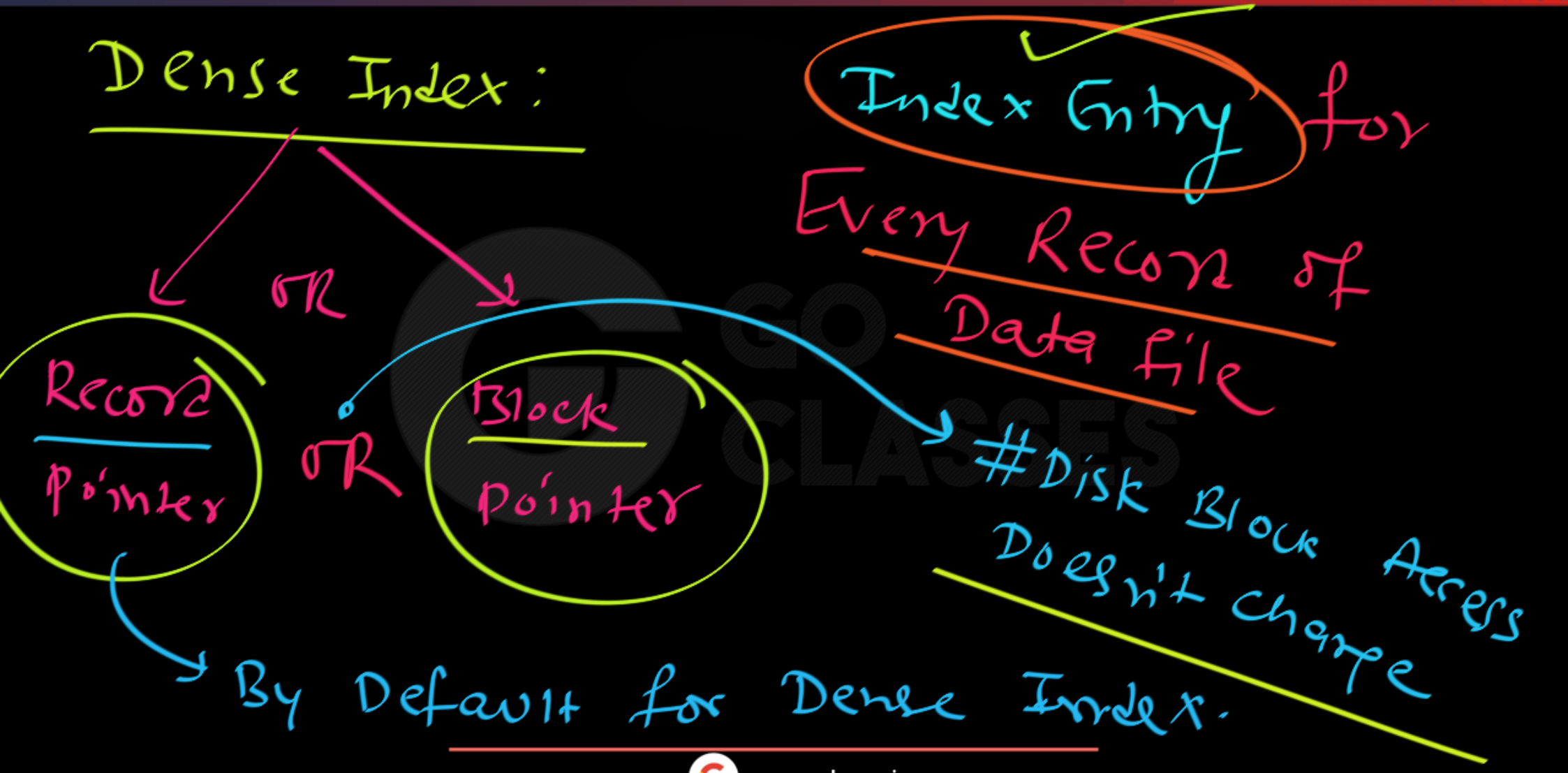
Physical order of Data file

Unordered

Secondary Index on Acc-no

A-101	R_2
A-102	R_1
A-201	R_3
A-215	R_6
A-217	R_4
A-222	R_5
A-305	R_7

Record addresses


Dense Index :
Index Entry for
Every Recon of
Data file
OR
Record
Pointer
OR
Block
Pointer
Disk Block Access
Doesn't change
By Default for Dense Index.

Sparse Index :

Index Entry for Some Records of Data file

Block Pointer

Data Access Always Happens Block by Block.

Assume this is Also physical order.

account_number	branch_name	balance
A-101	Downtown	500
A-102	Perryridge	400
A-201	Brighton	900
A-215	Mianus	700
A-217	Brighton	750
A-222	Redwood	700
A-305	Round Hill	350

Data file ordered by

Acc-no

key of Data file

Assume this is Also physical order.

Index I_1 on Acc_no → Primary Index

Index I_2 on B_name ⇒ SI on Non-key

account_number	branch_name	balance
A-101	Downtown	500
A-102	Perryridge	400
A-201	Brighton	900
A-215	Mianus	700
A-217	Brighton	750
A-222	Redwood	700
A-305	Round Hill	350

Index on Balance : SI on Non key

SI on Non-key

Assume this is Also physical order.

account_number	branch_name	balance
B_1 A-101	Downtown	500
B_1 A-102	Perryridge	400
B_2 A-201	Brighton	900
B_2 A-215	Mianus	700
B_3 A-217	Brighton	750
B_3 A-222	Redwood	700
B_4 A-305	Round Hill	350

SI on Balance :

Implementation I :

350	B_4
400	B_1
500	B_1
700	B_2, B_3
750	B_3
900	B_2

→ NOT preffered (variable length Index Records)

SI on Non-key

Assume this is Also physical order.

account_number	branch_name	balance
B_1 A-101	Downtown	500
B_1 A-102	Perryridge	400
B_2 A-201	Brighton	900
B_2 A-215	Mianus	700
B_3 A-217	Brighton	750
B_3 A-222	Redwood	700
B_4 A-305	Round Hill	350

SI on Balance :

Implementation 2 :

350	B_4
400	B_1
500	B_1
700	B_2
700	B_3
750	B_3
900	B_2

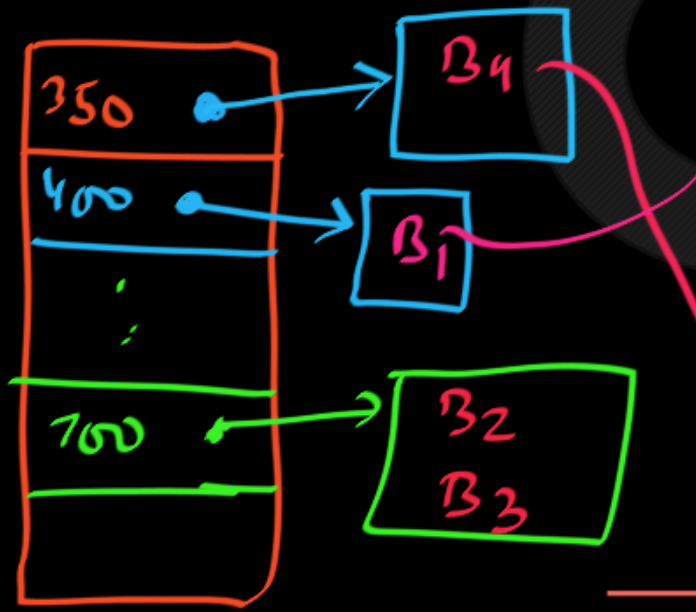
→ NOT preffered (Why ? we'll see in multilevel Indexes)

SI on Non-key

Assume this is Also physical order.

SI on Balance :

Implementation 3:



account_number	branch_name	balance
B_1 A-101	Downtown	500
B_1 A-102	Perryridge	400
B_2 A-201	Brighton	900
B_2 A-215	Mianus	700
B_3 A-217	Brighton	750
B_3 A-222	Redwood	700
B_4 A-305	Round Hill	350

by Default Implementation

Assume this Also the Physical order of Data file

Heap organization

A	B	D
a_1	b_1	d_1
a_1	b_1	d_2
a_1	b_2	d_1
a_2	b_2	d_1
a_3	b_1	d_2
a_2	b_2	d_2
a_2	b_2	d_3
a_1	b_1	d_4

Index 1: on D

Index 2: on B

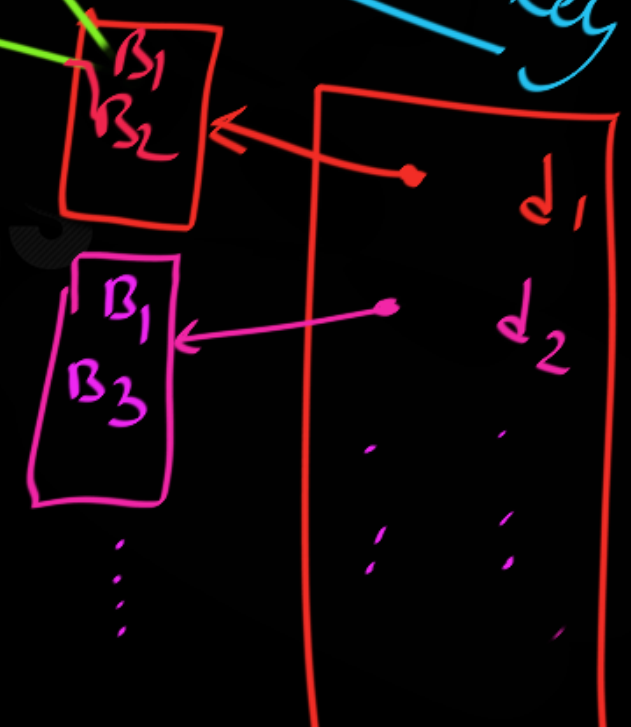
Index 3: on A

Secondary Indexes

Assume this Also the Physical order of Data file

Heap organization

	A	B	D
B_1	a_1	b_1	d_1
B_1	a_1	b_1	d_2
B_2	a_1	b_2	d_1
B_2	a_2	b_2	d_1
B_3	a_3	b_1	d_2
B_3	a_2	b_2	d_2
B_4	a_2	b_2	d_3
B_4	a_1	b_1	d_4



Q:

From DBMS point of view,
Index at the end of RaghuRama Krishnana book (Or any other standard book), is Secondary Index ????

SUBJECT INDEX

GO Classes

Data file is NOT sorted by Topic Name

Search key : Topic Name

Secondary Index

1NF, 430
2NF, 434
2PC, 628, 630
  blocking, 630
  with Presumed Abort, 631
2PL, 542
  distributed databases, 624
3NF, 432, 440, 443
3PC, 632
4NF, 447
5NF, 449
A priori property, 710
Abandoned privilege, 504
Abort, 525–526, 548, 556, 574, 584, 628
Abstract data types, 742
ACA schedule, 531
Access control, 8, 497–498
Access methods, 217
Application servers, 647
Architecture of a DBMS, 18
ARIES recovery algorithm, 571, 587
Armstrong's Axioms, 427
Array chunks, 693, 760
Arrays, 746
Assertions in SQL, 163
Association rules, 714, 716
  use for prediction, 718
  with calendars, 717
  with item hierarchies, 715
Asynchronous replication, 611, 620–621, 681
  Capture and Apply, 622
  change data table (CDT), 622
  conflict resolution, 621
  peer-to-peer, 621
  primary site, 621

Q:

Consider a Heap file (Do you remember what a Heap file is ???)

Can we create Secondary Index on a heap file ??

Q:

Consider a Heap file (Do you remember what a Heap file is ???)

Can we create Secondary Index on a heap file ?? ⇒ Yes.

NOTE:

So, to create Secondary Index on a file R:

R can be a Heap file OR a Sequential file...

Index on Unordered field of Data file, is called Secondary Index.

Q:

yes (3 SI possible simultaneously)

Can we create Secondary Index on the following File? (Assume the Order of recorded shown is the physical order of records)

If yes, then on which search key we should create Secondary Index??

Database Management System

10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

Q: Yes. → 3 SI Simultaneously Possible

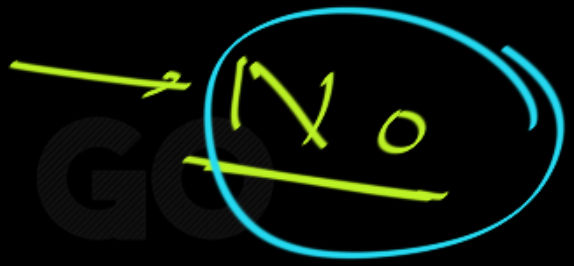
Can we create Secondary Index on the following File? (Assume the Order of recorded shown is the physical order of records)

If yes, then on which search key we should create Secondary Index??

76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
45565	Katz	Comp. Sci.	75000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000
12121	Wu	Finance	90000
76543	Singh	Finance	80000
32343	El Said	History	60000
58583	Califieri	History	62000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
33465	Gold	Physics	87000

Q:

Index on Non Candidate key is called Secondary Index ??



Q:

Index on unordered field of Data file is called Secondary Index ??

Yes

Q:

Consider a relation with at least two attributes, stored on a disk, either in sequential organization Or heap organization...

Can we ALWAYS create at least one secondary index??

Q:

Consider a relation with at least two attributes, stored on a disk, either in sequential organization Or heap organization...

Can we ALWAYS create at least one secondary index??

YES

Q:

How many Secondary Indexes can be created on a file (using Simple attribute) ??

Sequential

(#attribute - 1)

We only study Indexes on Simple attribute

Q:

How many Secondary Indexes can be created on a file (using Simple attribute) ??

Heap

We only study Indexes on Simple attribute

#attributes

Secondary Index on Key:

Sorted

Implementation

Dense

SI on key : → Always Dense

#Entries in Index = #Records in Data file

Secondary Index (key)

Can be built on ordered and also other type of files

Index attribute: non-ordering key field

Index entry:

value of the NOF V_i	pointer to the *record* with V_i as the NOF value

Index file: ordered file (sorted on NOF values)

No. of entries – same as the no. of *records* in the data file

Index file blocking factor $Bf_i = \lfloor B/(V+P_r) \rfloor$

(B: block size, V: length of the NOF,

P_r: length of a record pointer)

Index file blocks $= \lceil r/Bf_i \rceil$

(r – no. of records in the data file)

An Example

Index on this field : SI is key

Data file:

No. of records r = 90,000

Block size B = 4KB

Record length R = 100 bytes

[redacted]

NOF length V = 15 bytes

length of a record pointer P_r = 7 bytes

Key

An Example

Index on this field : SI is key

Data file:

No. of records r = 90,000

Record length R = 100 bytes

NOF length V = 15 bytes

Block size B = 4KB

#file Blocks

BF = $\lfloor 4096/100 \rfloor$ = 40,

b = $\lceil 90000/40 \rceil$ = 2250

length of a record pointer P_r = 7 bytes

Key

1. Index entry size = 15 + 7 = 22 B
2. Blocking factor of Index = # Index Records Per Block

Block size: 4 KB $\rightarrow 2^{10}$

Index Record size = 22 B

Index Blocking factor

$$\#\text{Index Records per Block} = \left\lfloor \frac{4096}{22} \right\rfloor = 186$$

(3) $\#\text{Index entries} = 90,000$

Block size: 4 KB $\rightarrow 2^{10}$

Index Record size = 22 B

$$\#\text{Index Records per Block} = \left\lfloor \frac{4096}{22} \right\rfloor = 186$$

(Index Blocking factor)

(4) $$\#\text{Index Blocks} : \left\lceil \frac{90,000}{186} \right\rceil = 484 \text{ Index Blocks}$$

(5) Access Cost using SI :

max = b = 2250

Avg = b/2 = 1125

$\lceil \log_2 I \rceil + 1 = 10$

→ 484

LOTS difference

(6) Access Cost with SI :

Can't Do Binary Search

file not sorted by search key

An Example

Data file:

No. of records r = 90,000 Block size B = 4KB

Record length R = 100 bytes $BF = \lfloor 4096/100 \rfloor = 40$,
$b = \lceil 90000/40 \rceil = 2250$

NOF length V = 15 bytes length of a record pointer $P_r = 7$ bytes

Index file :

No. of records $r_i = 90{,}000$ record length $= V + P_r = 22$ bytes

$BF_i = \lfloor 4096/22 \rfloor = 186$ No. of blocks $b_i = \lceil 90000/186 \rceil = 484$

Max no. of block accesses to get a record using the secondary index $1 + \lceil \log_2 b_i \rceil = 10$

Avg no. of block accesses to get a record without using the secondary index $b/2 = 1125$

A very significant improvement

NOTE:

Secondary Index is ALWAYS Dense Index.

So, in the index entry, second field can be Record pointer Or block pointer. (If record pointer and block pointer both are given in the question, take Record pointer for Dense Indexes)

17.1.3 Secondary Indexes

A **secondary index** provides a secondary means of accessing a data file for which some primary access already exists. The data file records could be ordered, unordered, or hashed. The secondary index may be created on a field that is a candidate key and has a unique value in every record, or on a nonkey field with duplicate values. The index is again an ordered file with two fields. The first field is of the same data type as some *nonordering field* of the data file that is an **indexing field**. The second field is either a *block* pointer or a *record* pointer. *Many* secondary indexes (and hence, indexing fields) can be created for the same file—each represents an additional means of accessing that file based on some specific field.

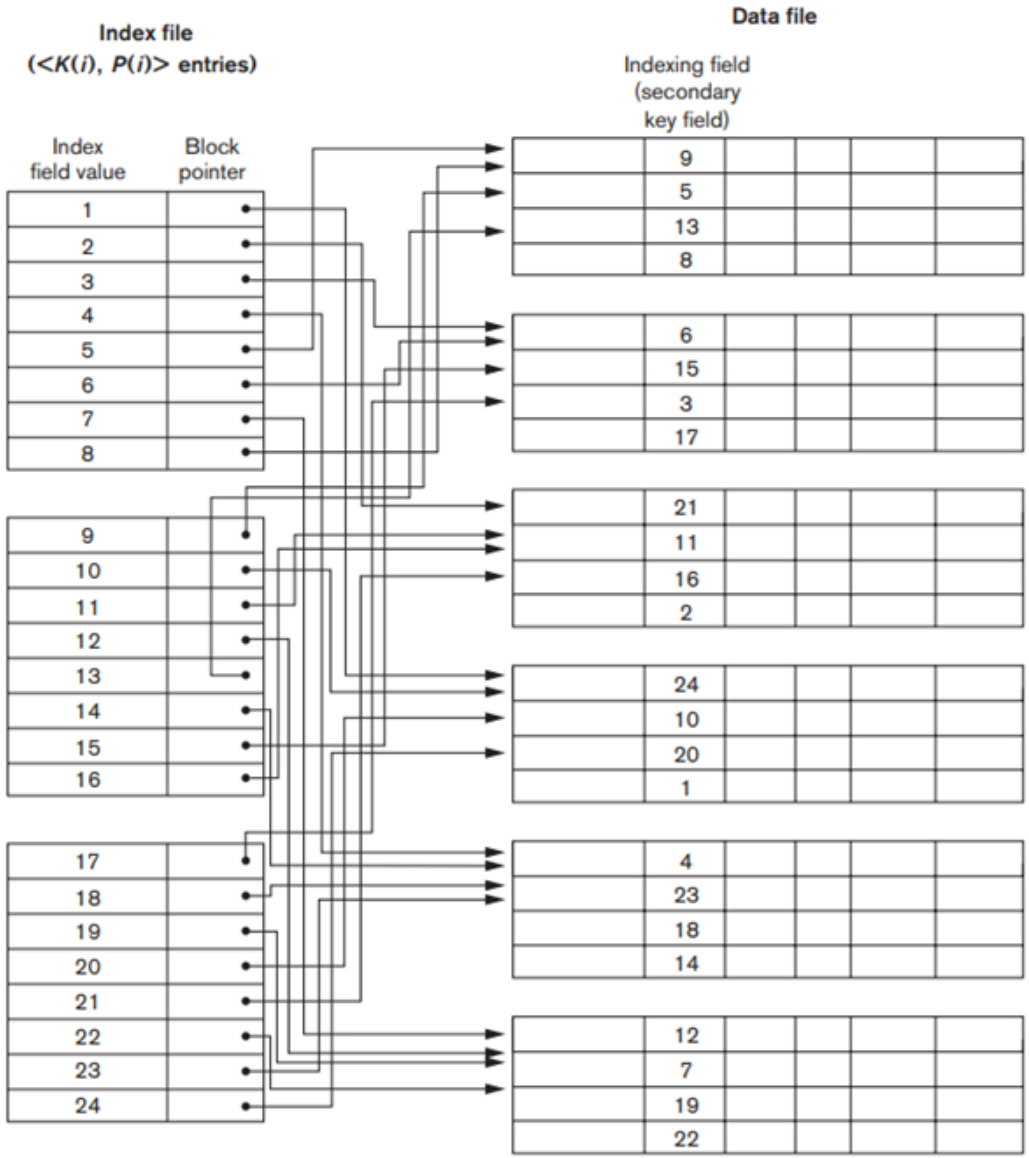
First we consider a secondary index access structure on a key (unique) field that has a *distinct value* for every record. Such a field is sometimes called a **secondary** key; in the relational model, this would correspond to any UNIQUE key attribute or to the primary key attribute of a table. In this case there is one index entry for *each record* in the data file, which contains the value of the field for the record and a pointer either to the block in which the record is stored or to the record itself. Hence, such an index is **dense**.

A secondary index usually needs more storage space and longer search time than does a primary index, because of its larger number of entries. However, the *improvement* in search time for an arbitrary record is much greater for a secondary index than for a primary index, since we would have to do a *linear search* on the data file if the secondary index did not exist. For a primary index, we could still use a binary search on the main file, even if the index did not exist. Example 3 illustrates the improvement in number of blocks accessed.

Example 3. Consider the file of Example 1 with $r = 300{,}000$ fixed-length records of size $R = 100$ bytes stored on a disk with block size $B = 4{,}096$ bytes. The file has $b = 7{,}500$ blocks, as calculated in Example 1. Suppose we want to search for a record with a specific value for the secondary key—a nonordering key field of the file that is $V = 9$ bytes long. Without the secondary index, to do a linear search on the file would require $b/2 = 7{,}500/2 = 3{,}750$ block accesses on the average. Suppose that we construct a secondary index on that *nonordering key* field of the file. As in Example 1, a block pointer is $P = 6$ bytes long, so each index entry is $R_i = (9 + 6) = 15$ bytes, and the blocking factor for the index is $bfr_i = \lfloor (B/R_i) \rfloor = \lfloor (4{,}096/15) \rfloor = 273$ index entries per block. In a dense secondary index such as this, the total number of index entries r_i is equal to the *number of records* in the data file, which is 300,000. The number of blocks needed for the index is hence $b_i = \lceil (r_i/bfr_i) \rceil = \lceil (300{,}000/273) \rceil = 1{,}099$ blocks.

A binary search on this secondary index needs $\lceil (\log_2 b_i) \rceil = \lceil (\log_2 1{,}099) \rceil = 11$ block accesses. To search for a record using the index, we need an additional block access to the data file for a total of 11 + 1 = 12 block accesses—a vast improvement over the 3,750 block accesses needed on the average for a linear search, but slightly worse than the 6 block accesses required for the primary index. This difference arose because the primary index was nondense and hence shorter, with only 28 blocks in length as opposed to the 1,099 blocks dense index here.

Figure 17.4
A dense secondary index (with block pointers) on a nonordering key field of a file.



Secondary Index on Non-Key: Various Implementations possible

Secondary Index on Non-Key:

Implementation 1: Like a Book Index

(NOT Good Implementation)

SUBJECT INDEX

1NF, 430
2NF, 434
2PC, 628, 630
  blocking, 630
  with Presumed Abort, 631
2PL, 542
  distributed databases, 624
3NF, 432, 440, 443
3PC, 632
4NF, 447
5NF, 449
A priori property, 710
Abandoned privilege, 504
Abort, 525–526, 548, 556, 574, 584, 628
Abstract data types, 742
ACA schedule, 531
Access control, 8, 497–498
Access methods, 217
Application servers, 647
Architecture of a DBMS, 18
ARIES recovery algorithm, 571, 587
Armstrong's Axioms, 427
Array chunks, 693, 760
Arrays, 746
Assertions in SQL, 163
Association rules, 714, 716
  use for prediction, 718
  with calendars, 717
  with item hierarchies, 715
Asynchronous replication, 611, 620–621, 681
  Capture and Apply, 622
  change data table (CDT), 622
  conflict resolution, 621
  peer-to-peer, 621
  primary site, 621

Implementation I :

↳ NOT preferred :

↳ Why? ⇒ Variable length Index Records

Secondary Index on Non-Key:

Implementation 2: Index entry for every record (NOT Good Implementation)

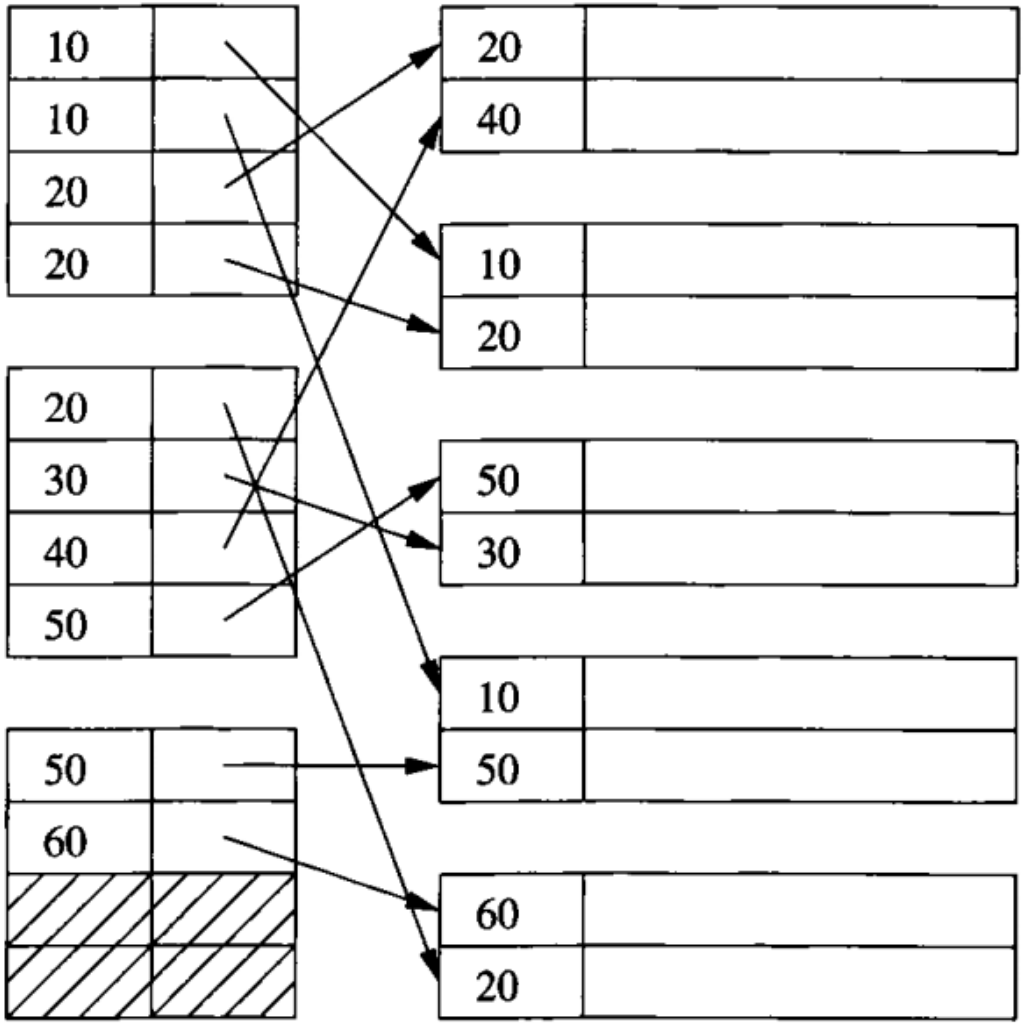
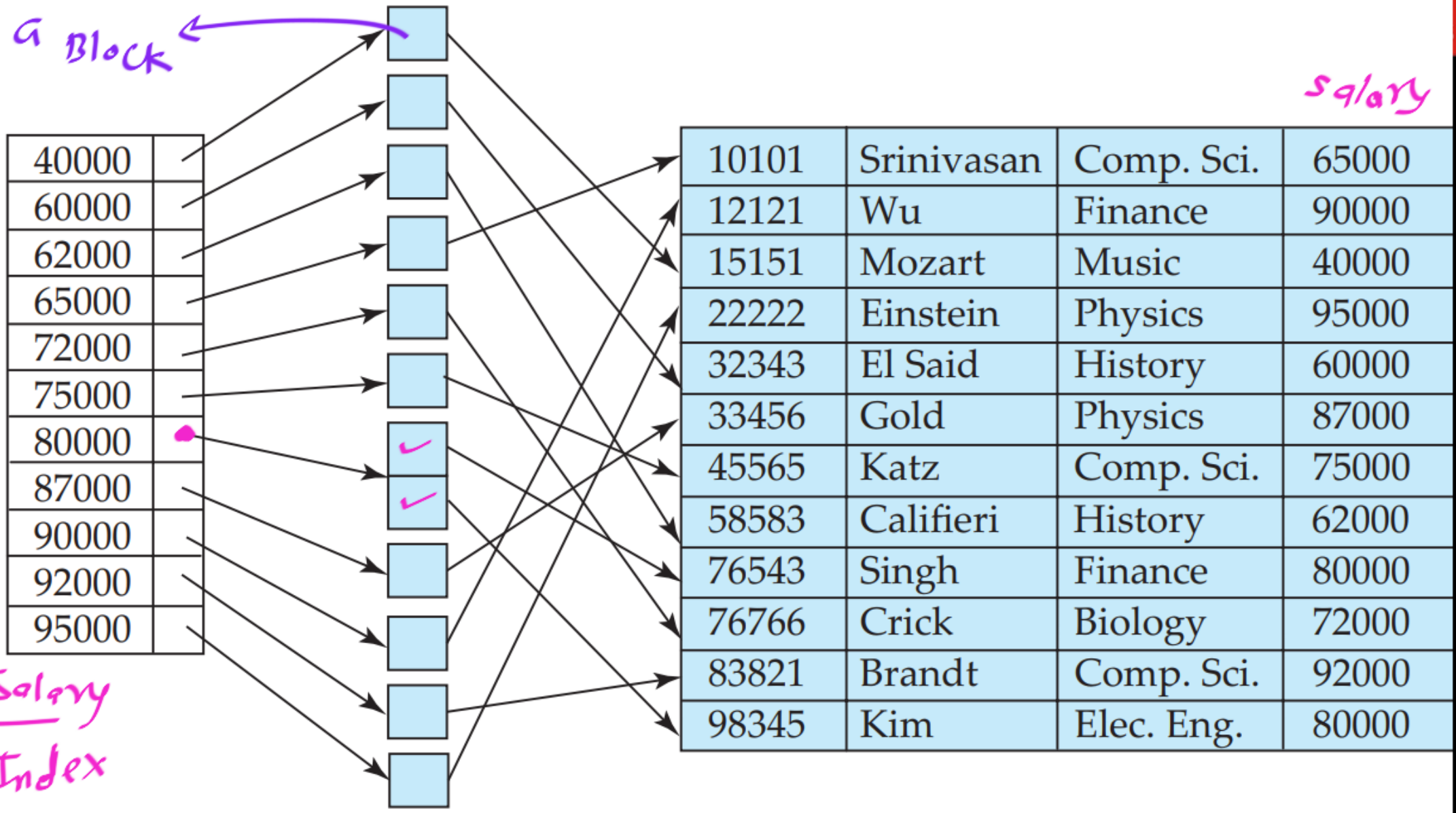


Figure 14.5: A secondary index

Secondary Index on Non-Key:

Implementation 3: A new level of pointers

(Good & Default Implementation)


a Block
Salary
40000
60000
62000
65000
72000
75000
80000
87000
90000
92000
95000
Salary
Index
10101 Srinivasan Comp. Sci. 65000
12121 Wu Finance 90000
15151 Mozart Music 40000
22222 Einstein Physics 95000
32343 El Said History 60000
33456 Gold Physics 87000
45565 Katz Comp. Sci. 75000
58583 Califieri History 62000
76543 Singh Finance 80000
76766 Crick Biology 72000
83821 Brandt Comp. Sci. 92000
98345 Kim Elec. Eng. 80000

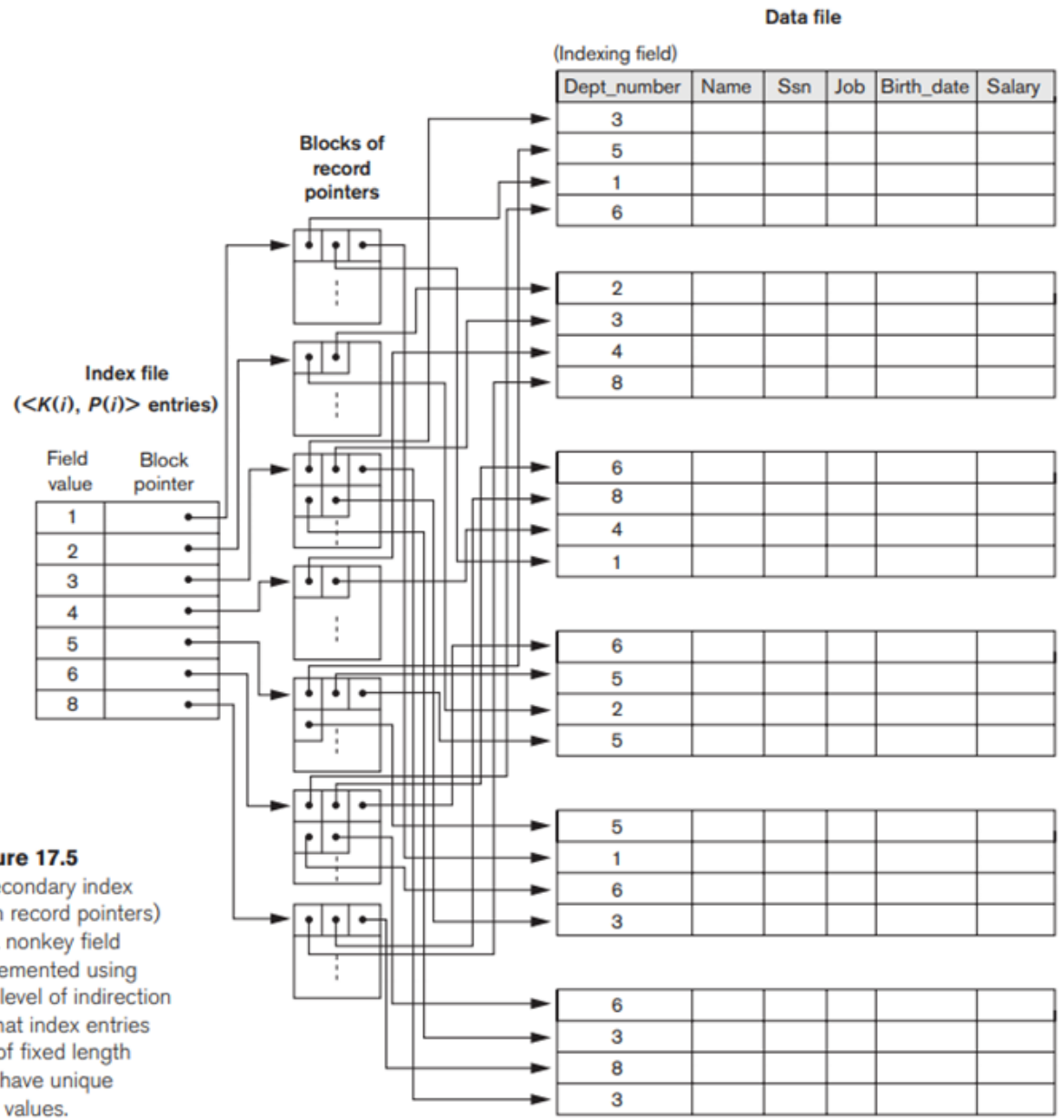


Figure 17.5
A secondary index (with record pointers) on a nonkey field implemented using one level of indirection so that index entries are of fixed length and have unique field values.

- Option 3, which is more commonly used, is to keep the index entries themselves at a fixed length and have a single entry for each *index field value*, but to create *an extra level of indirection* to handle the multiple pointers. In this scheme, the pointer $P(i)$ in index entry $<K(i), P(i)>$ points to a disk block, which contains a *set of record pointers*; each record pointer in that disk block points to one of the data file records with value $K(i)$ for the indexing field. If some value $K(i)$ occurs in too many records, so that their record pointers cannot fit in a single disk block, a cluster or linked list of blocks is used. This technique is illustrated in Figure 17.5. Retrieval via the

Secondary Index

Built on any non-ordering field (NOF) of a data file.

Case I: NOF is also a key (Secondary key)

value of the NOF V_i	pointer to the record with V_i as the NOF value

Case II: NOF is not a key: two options

(1)

value of the NOF V_i	pointer(s) to the record(s) with V_i as the NOF value

(2)

value of the NOF V_i	pointer to a block that has pointer(s) to the record(s) with V_i as the NOF value

Remarks:

(1) index entry – variable length record

(2) index entry – fixed length – One more level of indirection

Prof P Sreenivasa Kumar
Department of CS&E, IITM

Next Topic:

Ordered Indexes

Practice

Q 1: Consider relation R(A,B,C,D)
Assume A is the Key. File R stored in 100 Blocks.

SELECT * FROM R WHERE a=10;

No Index. File R uses Heap Organization.

Number of Disk Block Access needed??

Q 1: Consider relation R(A,B,C,D)

Assume A is the Key. File R stored in 100 Blocks.

SELECT * FROM R WHERE a=10;

Search based on A.

↳ at most record in o/p

Every field is unordered.

No Index. File R uses Heap Organization.

Number of Disk Block Access needed??

Worst: 100 Blocks ; Avg : b/2 = 50 Blocks

Q 2: Consider relation R(A,B,C,D)
Assume A is the Key. File R stored in 100 Blocks.

SELECT * FROM R WHERE a=10;

No Index. File R is Ordered by A.

Number of Disk Block Access needed??

Q 2: Consider relation R(A,B,C,D)

Assume A is the Key. File R stored in 100 Blocks.

↳ at most one record with A = 10

SELECT * FROM R WHERE a=10;

search based on A

↳ sequential file

No Index. File R is Ordered by A.

Number of Disk Block Access needed??

Binary search: $\lceil \log_2 100 \rceil = 7$ Blocks (worst case)

Q 3: Consider relation R(A,B,C,D)

Assume A is the Key.File R needs 100 Blocks.

SELECT * FROM R WHERE a=10;

No Index. File R is Ordered by B.

Number of Disk Block Access needed??

Q 3: Consider relation R(A,B,C,D)

Assume A is the Key. File R needs 100 Blocks.

SELECT * FROM R WHERE A = 10; } Searching on A

No Index. File R is Ordered by B. ⇒ Sequential by B

Number of Disk Block Access needed??

Can NOT use Binary search; Avg = 50; worst = 100 Blocks

Q 4:

Consider relation R(A,B,C,D)

Assume A, B are two Keys.

File R is Ordered by A.

Index on A is ??

Q 4:

Consider relation R(A,B,C,D)

Assume A, B are two Keys.

File R is Ordered by A.

Index on A is ?? Primary Index

Ordering key field of R

on A

Q 5:

Consider relation R(A,B,C,D)

Assume A, B are two Keys.

File R is Ordered by A.

Index on B is ??

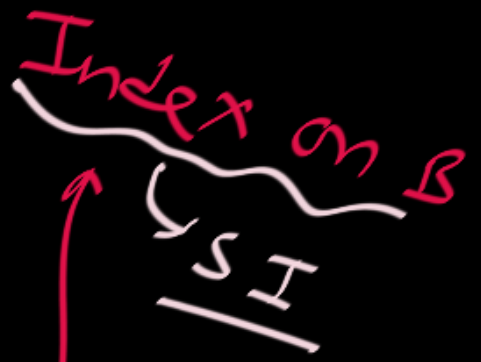
Q 5:

Consider relation R(A,B,C,D)

Assume A, B are two Keys.

File R is Ordered by A. ⇒ In file R, B is Unordered

Index on B is ?? ⇒ SI (on key)



Q 6:

Consider relation R(A,B,C,D)
Assume A, B are two Keys.
File R is Ordered by A.

Index on C is ??

Q 6:

Sorted ← → Unsorted

Consider relation R(A,B,C,D)

Assume A, B are two Keys.

File R is Ordered by A. ✓

Index on C is ?? SI on Non-key

Q 7:

Consider relation R(A,B,C,D)

Assume A, B are two Keys.

File R is Ordered by C.

Index on C is ??

Q 7: Data file → sorted

Consider relation R(A,B,C,D)

Assume A, B are two Keys.

File R is Ordered by C. → ordering field → Non-key

Index on C is ?? ⇒ Clustering Index (CI)

Q 8:

Consider relation R(A,B,C,D)
Assume A, B are two Keys.
File R is Ordered by C.

Index on A is ??

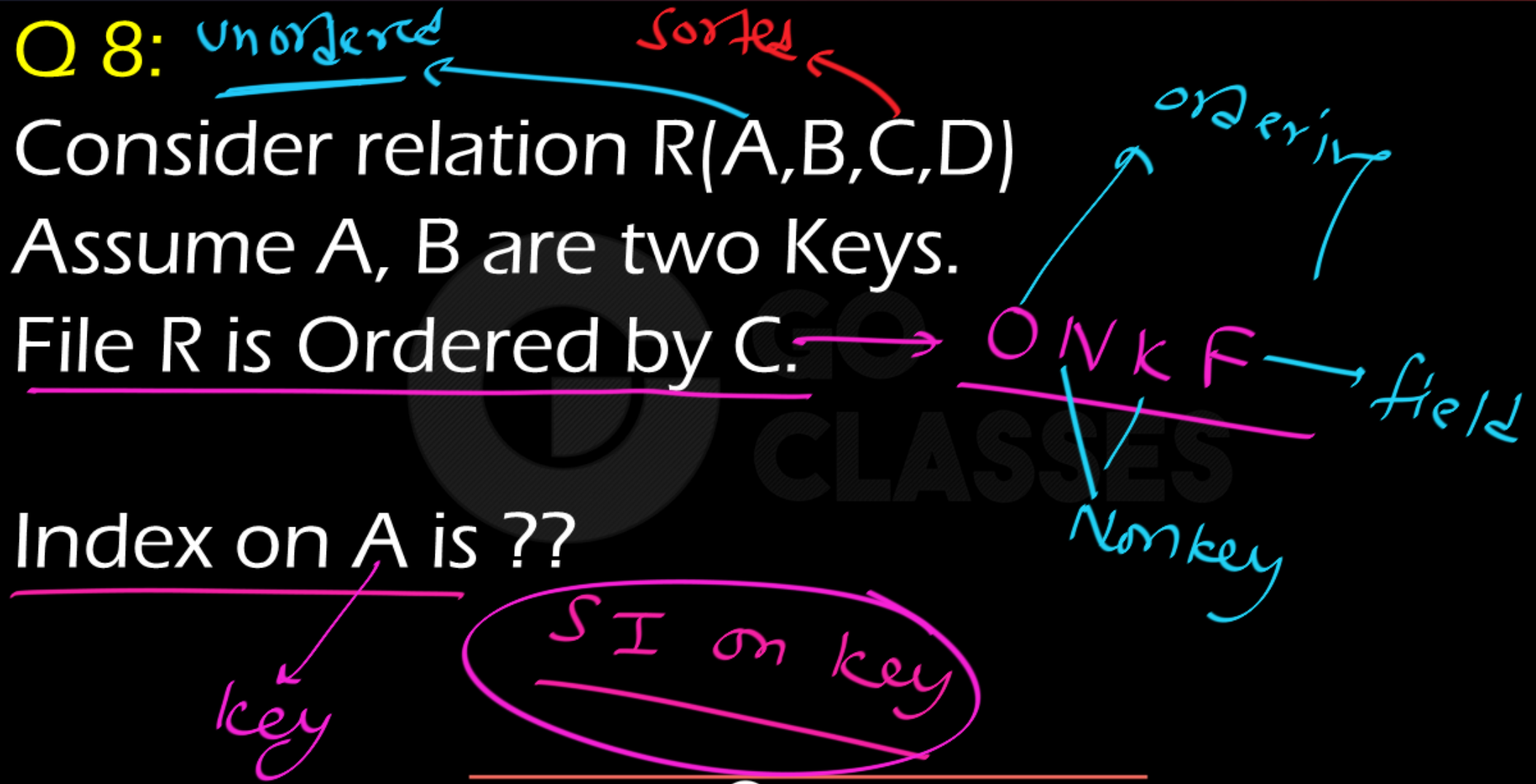
Q 8:

Consider relation R(A,B,C,D)

Assume A, B are two Keys.

File R is Ordered by C.

Index on A is ??



Q 9:

Consider relation R(A,B,C,D)
Assume A, B are two Keys.
File R is Ordered by C.

Index on D is ??

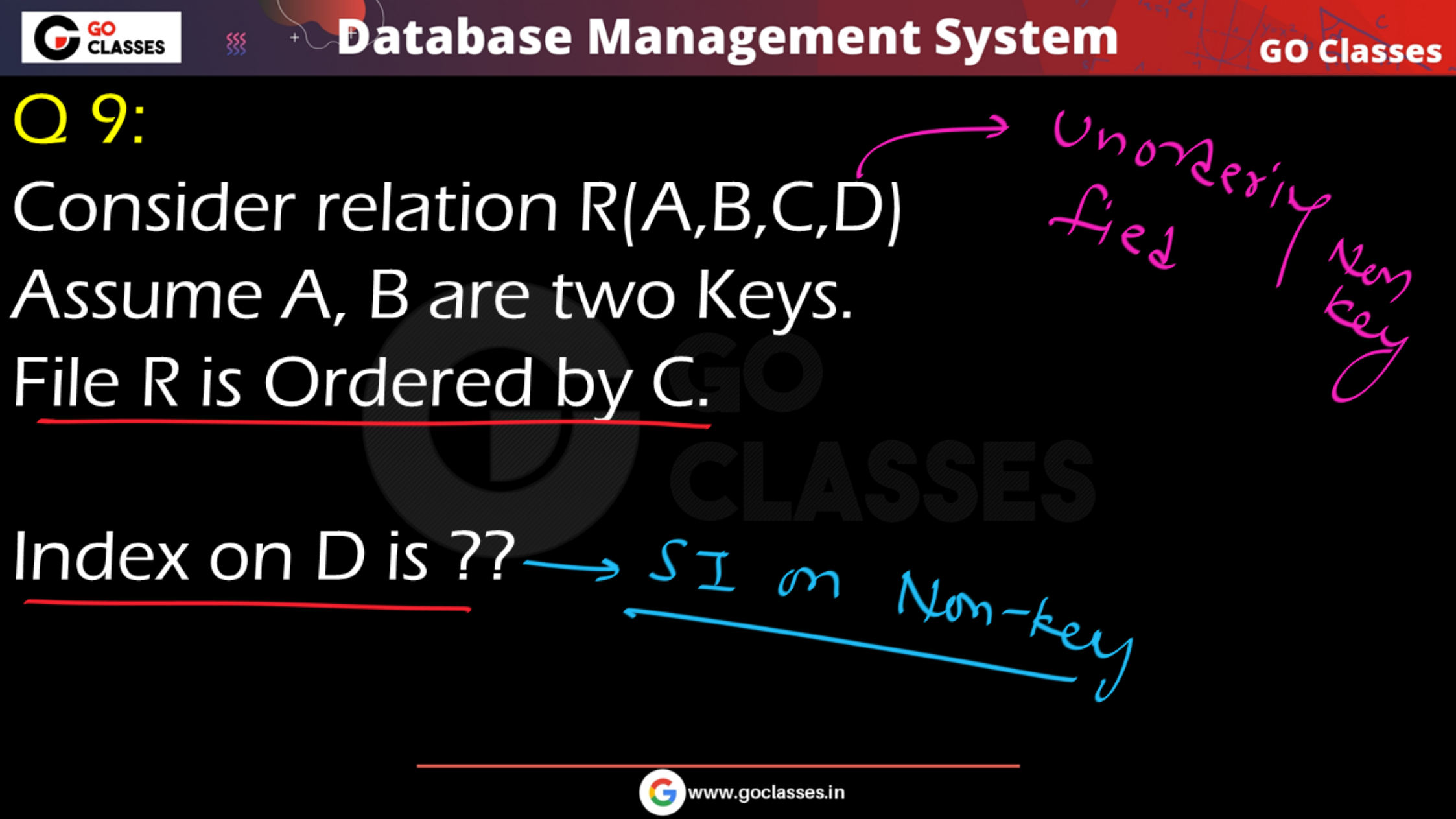
Q 9:

Consider relation R(A,B,C,D)

Assume A, B are two Keys.

File R is Ordered by C.

Index on D is ??



Q 10:

Consider relation R(A,B,C,D)

Assume A, B are two Keys.

File R is Heap File i.e. NO Field is Ordered in R.

Index on A is ??

Q 10:

Consider relation R(A,B,C,D)

unordered

Assume A, B are two Keys.

File R is Heap File i.e. NO Field is Ordered in R.

Index on A is ?? ⇒ SI on key

Q 11:

Consider relation R(A,B,C,D)

Assume A, B are two Keys.

File R is Heap File i.e. NO Field is Ordered in R.

Index on D is ??

Q 11:

Consider relation R(A,B,C,D)

Assume A, B are two Keys.

File R is Heap File i.e. NO Field is Ordered in R.

Index on D is ?? SI on Non-key

Q 12: Consider relation R(A,B,C,D)
Assume A, B are two Keys.
File R is sequentially ordered by A.

How many Indexes can we create (using single attribute search key) simultaneously??
Mention their types.

Q 12: Consider relation R(A,B,C,D)
Assume A, B are two Keys.
File R is sequentially ordered by A.

How many Indexes can we create (using single attribute search key) simultaneously??
Mention their types. ⇒ 4 Indexes

Data file R sorted on key A.

Index on A : PI

" " B : SI on key

" " C : SI " Nonkey

" " D : SI " "

Q 13: Consider relation R(A,B,C,D)
Assume A, B are two Keys.
File R is sequentially ordered by C.

How many Indexes can we create (using single attribute search key) simultaneously??
Mention their types.

Q 13: Consider relation R(A,B,C,D)
Assume A, B are two Keys.
File R is sequentially ordered by C.

4

How many Indexes can we create (using single attribute search key) simultaneously??
Mention their types.

R ordered by Non-key C :

Index on A : SI on key
" " B : SI " "
" " C : → CI
" " D : SI on Non-key

Q 14: Consider relation R(A,B,C,D)
Assume A, B are two Keys.
File R is a Heap file i.e. NO Field is Ordered in R.

How many Indexes can we create (using single attribute search key) simultaneously??
Mention their types.

Q 14: Consider relation R(A,B,C,D)

Assume A, B are two Keys.

File R is a Heap file i.e. NO Field is Ordered in R.

How many Indexes can we create (using single attribute search key) simultaneously?? Mention their types.

Q 16: Consider relation R.

We create an Index I on R. The (Physical) Order of search key in I is same or almost same as order of R.

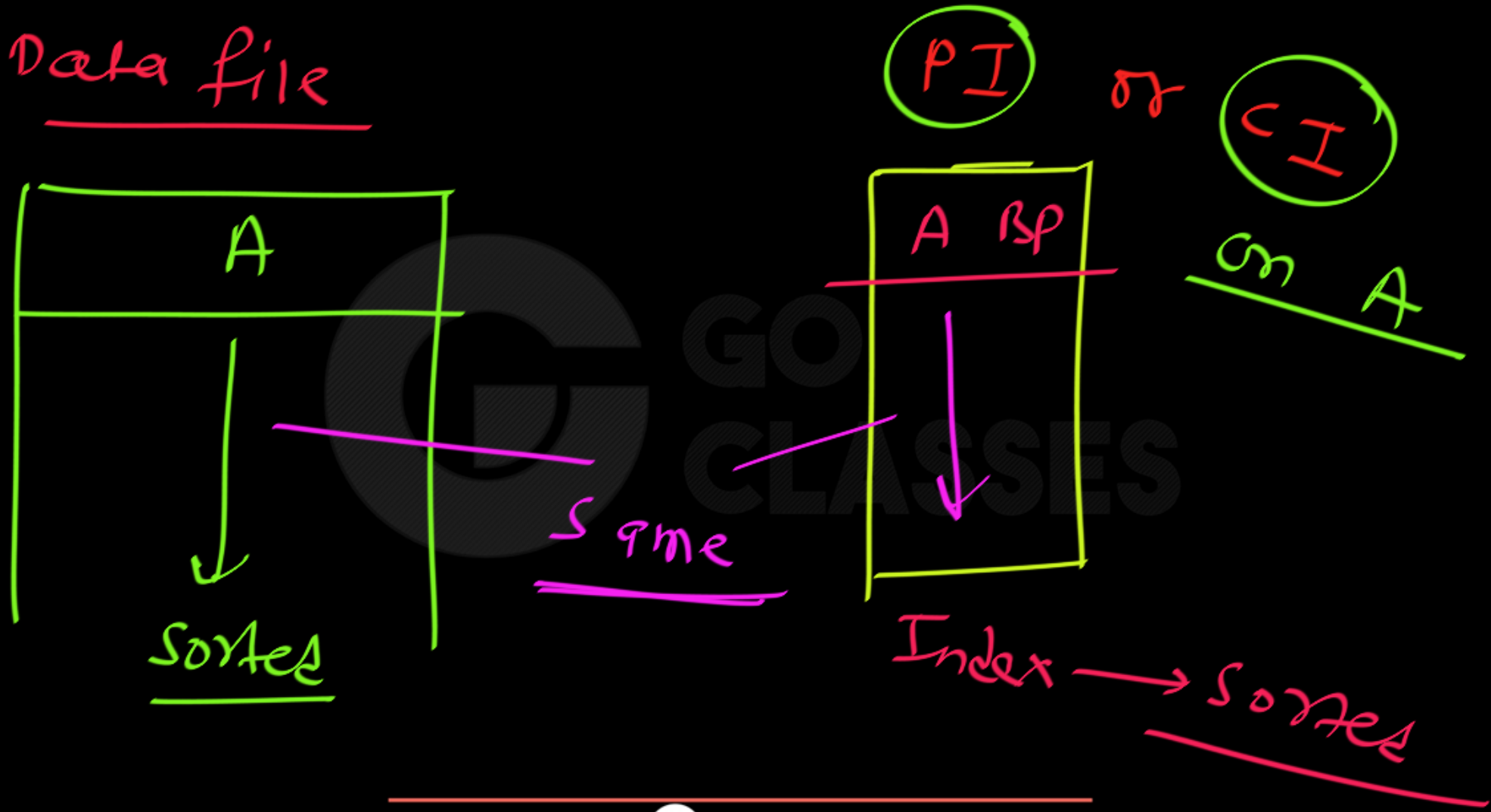
Then I is?

Q 16: Consider relation R.

We create an Index I on R. The (Physical) Order of search key in I is same or almost same as order of R.

Then I is?

PI OR CI


Data file
A
Sorted
Same
PI
or
CI
A Bp
Sm A
Index → Sorted

Q 17: Consider relation R.

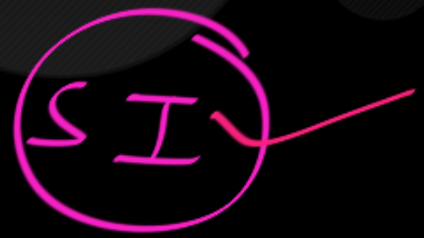
We create an Index I on R. The (Physical) Order of search key in I is different from the order of R.

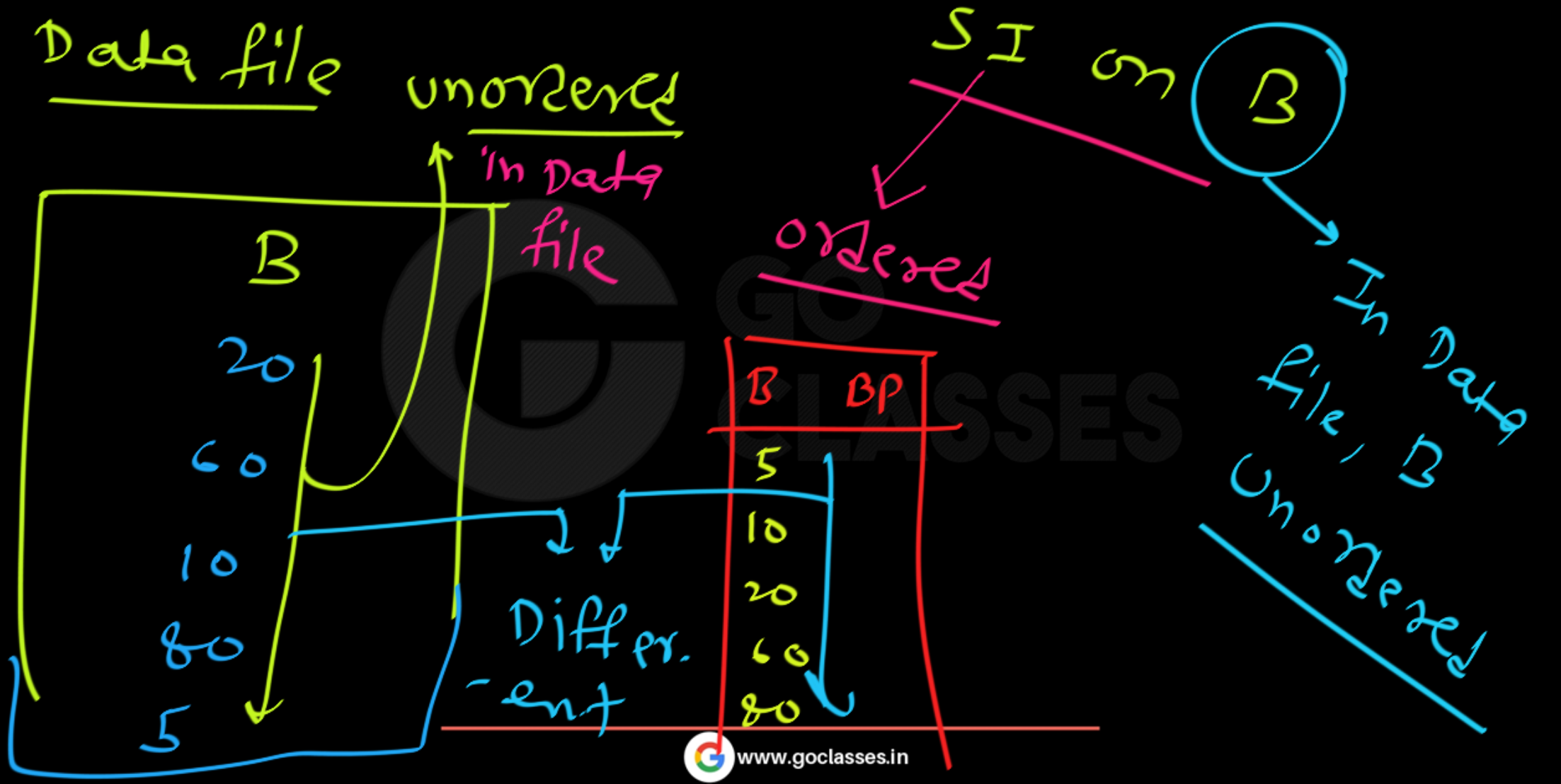
Then I is?

Q 17: Consider relation R.

We create an Index I on R. The (Physical) Order of search key in I is different from the order of R.

Then I is?




Data file
unordered
in Data file
B
20
60
10
80
5
Diff er. ent
SI on B
ordered
B
BP
5
10
20
60
80
In Data file, B Unordered

Q 18: GATE CSE 2008

3.7.5 Indexing: GATE CSE 2008 | Question: 16, ISRO2016-60 top https://gateoverflow.in/414

A clustering index is defined on the fields which are of type

A. non-key and ordering
B. non-key and non-ordering
C. key and ordering
D. key and non-ordering

https://gateoverflow.in/414/gate-cse-2008-question-16-isro2016-60

Q 18: GATE CSE 2008

3.7.5 Indexing: GATE CSE 2008 | Question: 16, ISRO2016-60 top https://gateoverflow.in/414

A clustering index is defined on the fields which are of type

A. non-key and ordering
B. non-key and non-ordering
C. key and ordering
D. key and non-ordering

https://gateoverflow.in/414/gate-cse-2008-question-16-isro2016-60

Q 19: GATE CSE 2013

GATE CSE 2013 | Question: 15

asked in Databases Sep 23, 2014

12,644 views

42

An index is clustered, if

A. it is on a set of fields that form a candidate key

B. it is on a set of fields that include the primary key

C. the data records of the file are organized in the same order as the data entries of the index

D. the data records of the file are organized not in the same order as the data entries of the index

SI

PI or RSI

PI or SI

https://gateoverflow.in/1437/gate-cse-2013-question-15

Q 20: GATE CSE 2015

3.7.9 Indexing: GATE CSE 2015 Set 1 | Question: 24 top https://gateoverflow.in/8222

A file is organized so that the ordering of the data records is the same as or close to the ordering of data entries in some index. Than that index is called

A. Dense
B. Sparse
C. Clustered
D. Unclustered

https://gateoverflow.in/8222/gate-cse-2015-set-1-question-24

Q 20: GATE CSE 2015

3.7.9 Indexing: GATE CSE 2015 Set 1 | Question: 24 top https://gateoverflow.in/8222

A file is organized so that the ordering of the data records is the same as or close to the ordering of data entries in some index. Than that index is called

A. Dense
B. Sparse
C. Clustered
D. Unclustered

CI OR PI

Secondary Index

https://gateoverflow.in/8222/gate-cse-2015-set-1-question-24

Q 20: GATE CSE 2015

3.7.9 Indexing: GATE CSE 2015 Set 1 | Question: 24 top https://gateoverflow.in/8222

Index entry for some Records of Data file

Than that index is called

A. Dense
B. Sparse
C. Clustered
D. Unclustered

https://gateoverflow.in/8222/gate-cse-2015-set-1-question-24

Indexing is a topic which many students think they understand BUT they lack the CRYSTAL Clarity.

We have studied in a CRYSTAL Clear Way, like we do ALWAYS in GO Classes.

Do you have CRYSTAL Clarity about All the types of Indexes now??

Tell me in Comments.

Q 21: CSIE @ NTU university question

How many clustered indexes can you create on a file?

Q 21: CSIE @ NTU university question

How many clustered indexes can you create on a file?

→ At most one

Q 22: CSIE @ NTU university question

Would you always be able to create at least one clustered index for a file?

Q 22: CSIE @ NTU university question

Would you always be able to create at least one clustered index for a file?

No

file is Heap file OR file is sequential by key

No Clustering Index possible

Q 23: True/False ??

A clustered index is one in which the (search key) order of the entries in the index corresponds to the order of the data records themselves in the underlying file for which the index was constructed.

Q 23: True/False ??

A clustered index is one in which the (search key) order of the entries in the index corresponds to the order of the data records themselves in the underlying file for which the index was constructed.

Navathe OR RR : Primary → 0 k f

Clustering → 0 N k F

Korth → Primary OR Clustering

Same

Q 24: True/False ??

It is possible to create several clustered indexes on a given data file.

It is possible to create several unclustered indexes on a given data file.

Q 24: True/False ??

Secondary

It is possible to create several clustered indexes on a given data file. False

It is possible to create several unclustered indexes on a given data file. True

Q 25: True/False ??

An index can only be created on a unique field or field(s), i.e., on a candidate key.

Q 25: True/False ??

An index can only be created on a unique field or field(s), i.e., on a candidate key.

false

Q 26:

What is a record id (record address)?

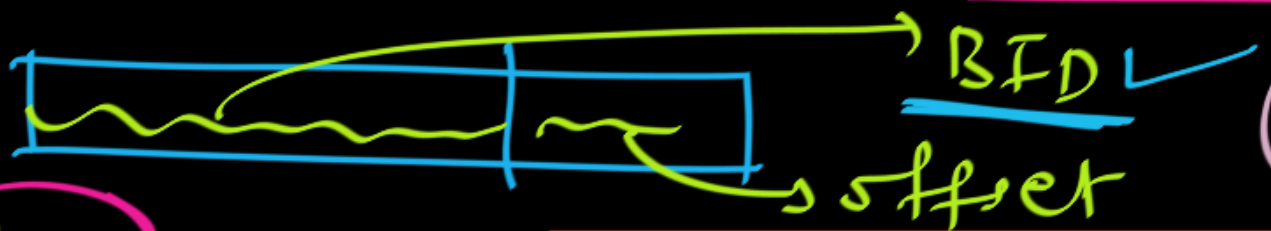
Given a record's id, how many I/Os are needed to fetch it into main memory?

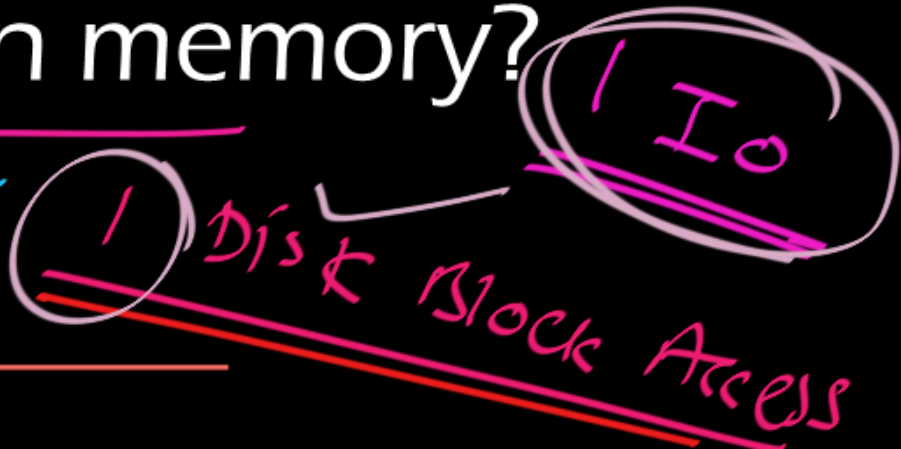
Q 26: RID = BID + offset within the Block

Record Address = Block Address + offset

What is a record id (record address)?

Given a record's id, how many I/Os are needed to fetch it into main memory?





A record id, or rid for short, is a unique identifier for a particular record in a set of records. An rid has the property that we can identify the disk address of the page containing the record by using the rid. The number of I/O's required to read a record, given a rid, is therefore 1 I/O.

Q 27:

Consider a Heap file (Do you remember what a Heap file is ???)

Can we create a Clustering Index on a heap file ??

Q 27:

Consider a Heap file (Do you remember what a Heap file is ???)

Can we create a Clustering Index on a heap file ??

Ans: NO. File must be Physically Ordered by a Non-key attribute.

Q 28:

How many Clustering Indexes can be created on a file ?

Q 28:

How many Clustering Indexes can be created on a file ?

Ans: At Most One

Q 29:

On a file, Can we create both Primary as well as Clustering Index ??

Q 29:

On a file, Can we create both Primary as well as Clustering Index ??

Ans: NO. At most one of them.

Q 29:

On a file, Can we create Primary OR Clustering Index ??

Ans: NO. At most one of them.

Some thought-provoking Questions...

THINK...

Q 30: True/False ??

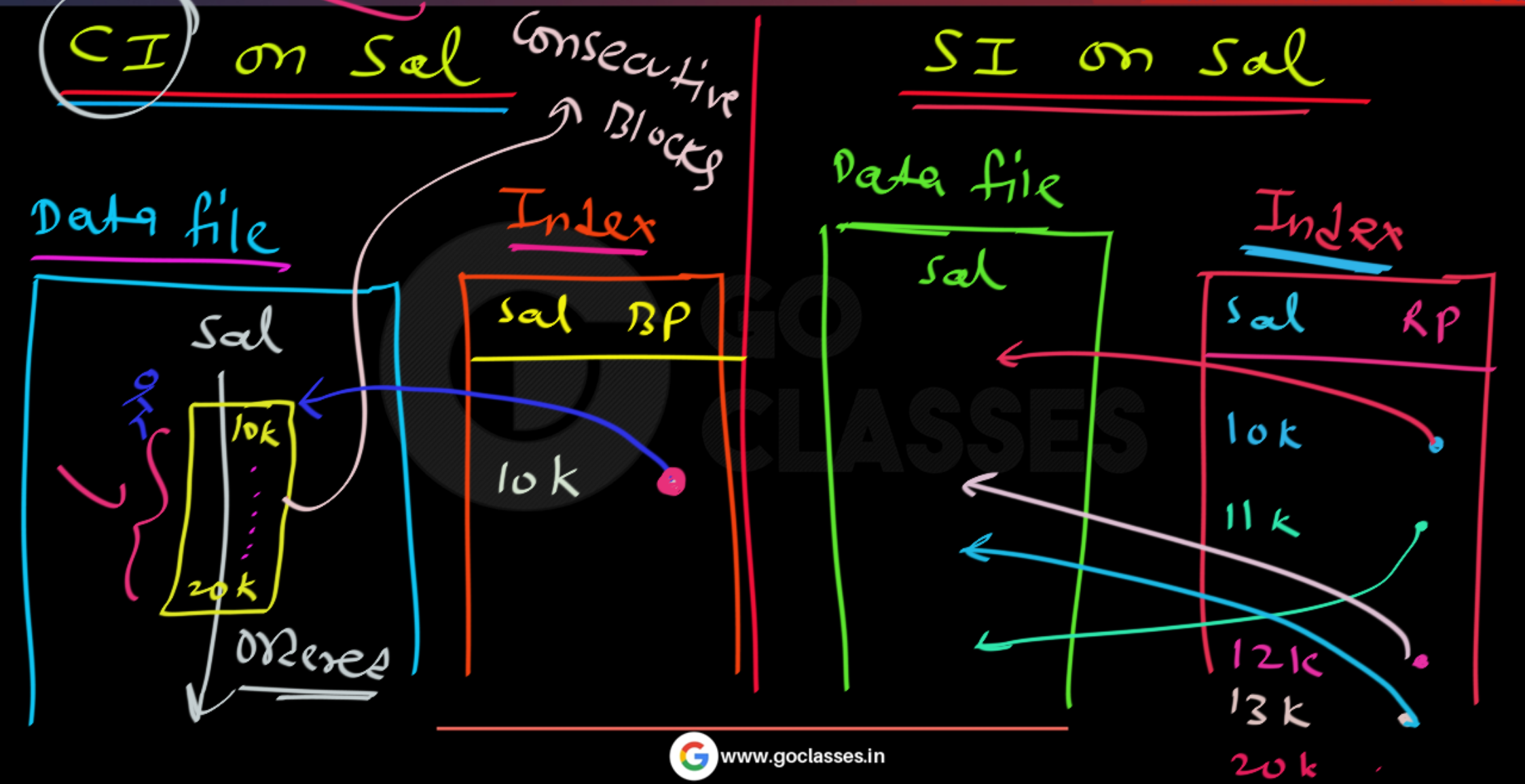
Processing a range query (e.g., salary BETWEEN 10000 AND 20000) using a clustered index is always faster than using an unclustered index.

Q 30: True/False ??

Processing a range query (e.g., salary BETWEEN 10000 AND 20000) using a clustered index is always faster than using an unclustered index.

Records with Sal : 15k to 20k

Data


CI on Sal
Consecutive Blocks
Data file
Sal
10k
20k
Index
Sal RP
10k
SI on Sal
Data file
Sal
Index
Sal RP
10k
11k
12k
13k
20k

Q 30: True/False ??

Processing a range query (e.g., salary BETWEEN 10000 AND 20000) using a clustered index is always faster than using an unclustered index.

If CI on Sal then Range Query records are found in Consecutive Blocks

Some thought-provoking Questions...

Is Index ALWAYS Helpful?

Q 31: True/False ??

Consider a relation stored as a randomly ordered file(i.e. Heap Organization) for which the only index is an unclustered(secondary) index on a field called sal.

If you want to retrieve all records with sal > 20, is using the index always the best alternative?

Q 31: True/False ??

Consider a relation stored as a randomly ordered file(i.e. Heap Organization) for which the only index is an unclustered(secondary) index on a field called sal.

If you want to retrieve all records with sal > 20, is using the index always the best alternative?

Exercise 8.3 Consider a relation stored as a randomly ordered file for which the only index is an unclustered index on a field called *sal*. If you want to retrieve all records with $sal > 20$, is using the index always the best alternative? Explain.

Answer 8.3 No. In this case, the index is unclustered, each qualifying data entry could contain an rid that points to a distinct data page, leading to as many data page I/Os as the number of data entries that match the range query. In this situation, using index is actually worse than file scan.

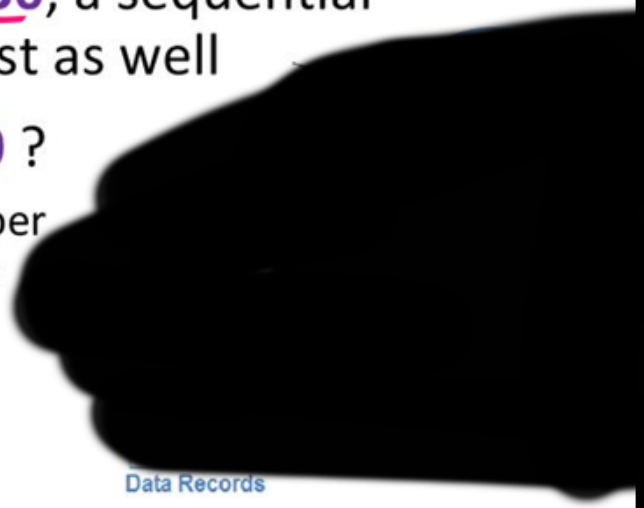
Is Index always helpful ?

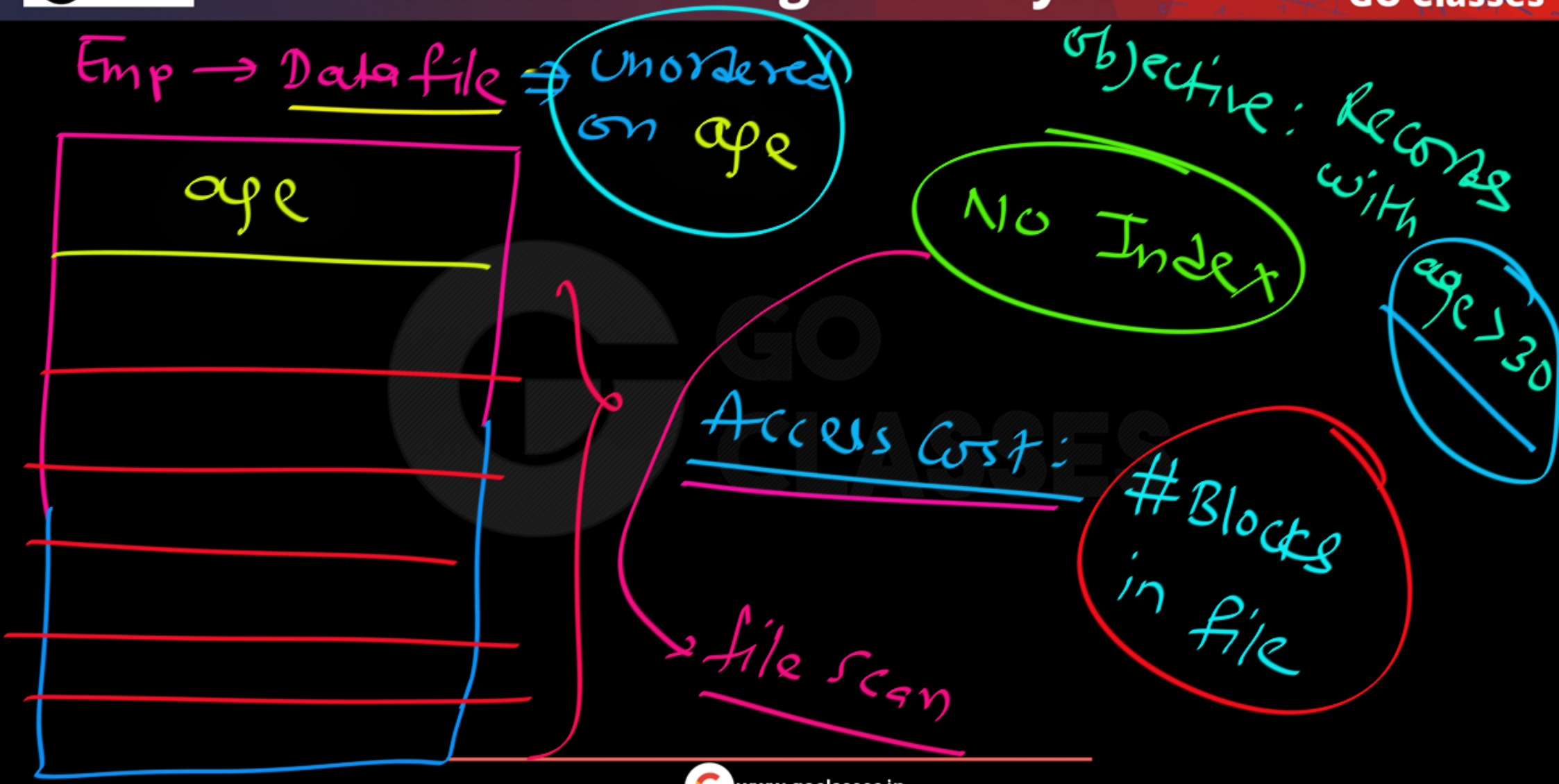
No

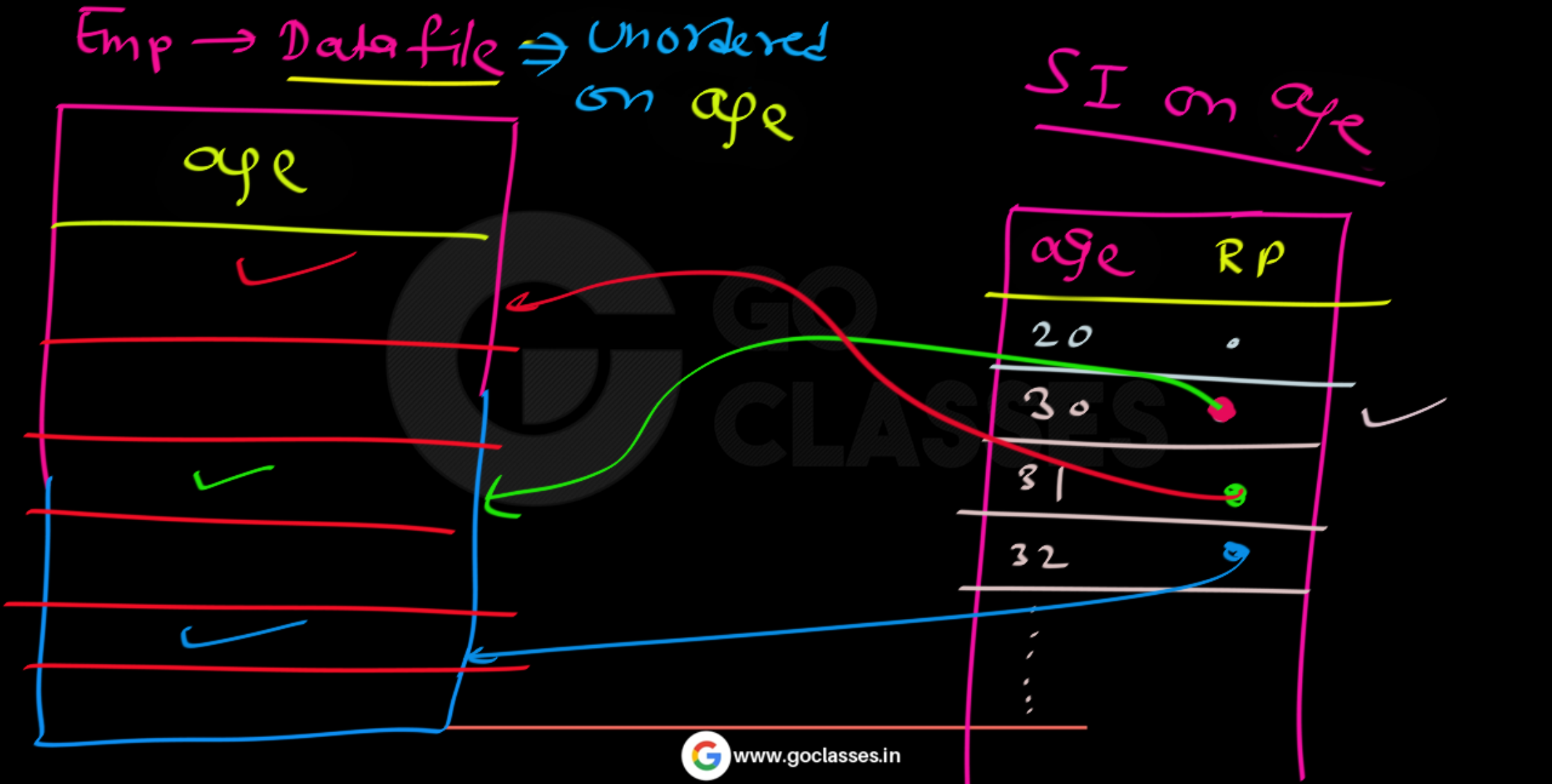
```
SELECT  E.dno
FROM    Emp E
WHERE  E.age>30
```

What is the selectivity of the condition ?

- If **most employees are older than 30**, a sequential scan of the relation would do almost as well
- What if **only 10% are older than 30** ?
  - **Unclustered index**: Performing one I/O per qualifying tuple could be more expensive than a sequential scan
  - **Clustered index**: This is a good option




Emp → Datafile
Unordered on age
age
Objective: Records with age > 30
No Index
Access Cost:
Blocks in file
file scan

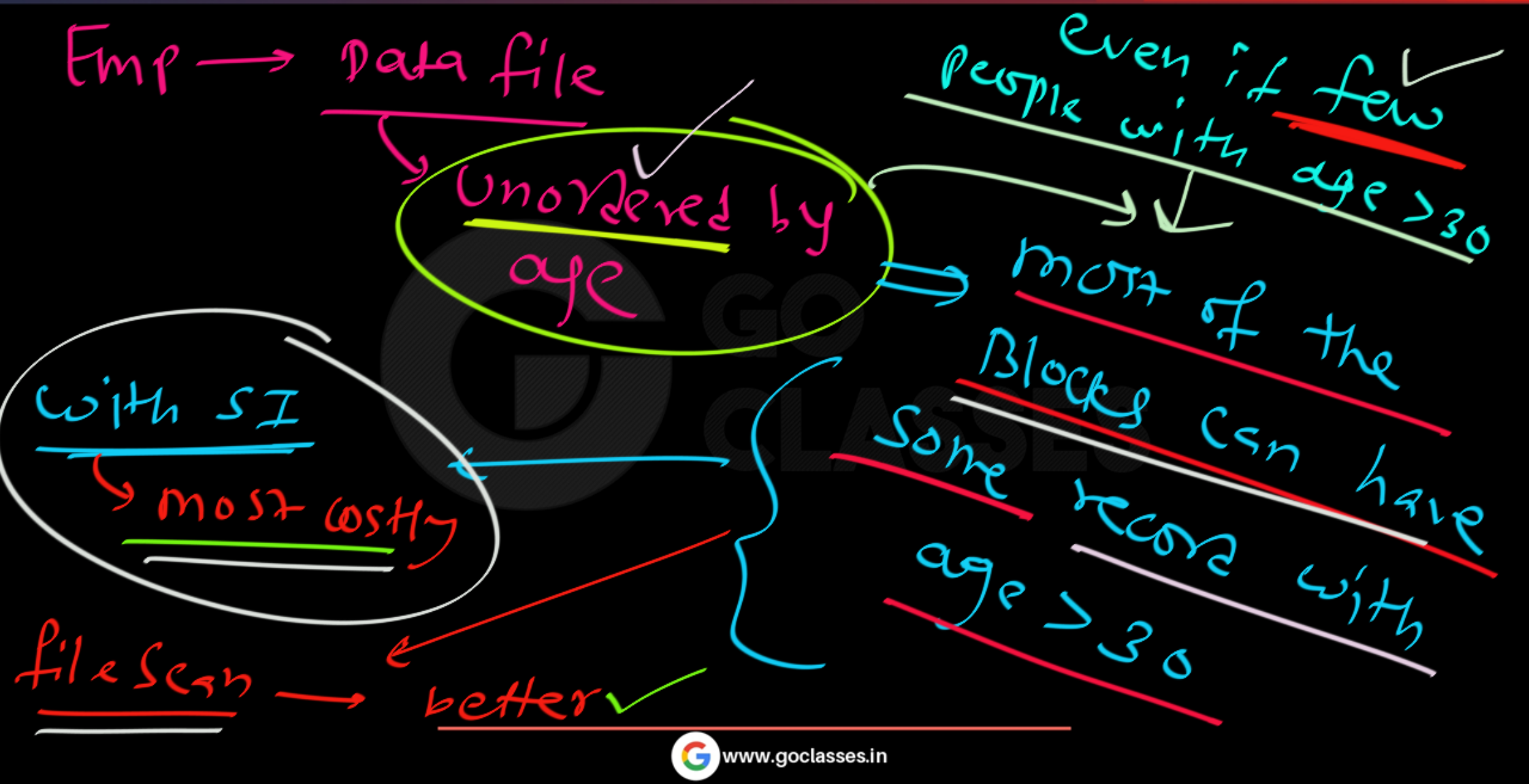

Emp → Data file ⇒ Unordered on age
age
SI on age
age
RP
20
30
31
32

Age > 30

Assume Almost everyone has Age > 30

{ with SI on age :

file Blocks + # Index Blocks


Emp → Data file
Unordered by age
Even if few people with age > 30
most of the Blocks can have some records with age > 30
with SI
most costly
fileScan → better

Conclusion:

SI is Useful : When we want a particular Record which can exist in very few number of Blocks.

Point Queries

Ex: People with age = 50 ⟹ SI on Age

Very Good.

Conclusion:

SI is Useful : when we don't have to access too many Data Blocks.

Ex: People with age = 50 ⟹ SI on Age Very Good.

Conclusion:

SI is Burden: when we have to access too many Data Blocks.

Eg: Range Queries

In Data file Age unsorted

Eg: People with 30 < age < 50 ⇒ SI on age

↳ Not useful

Q 32: True/False ??

Processing a range query using an unclustered index is always faster than using a file scan.

Q 32: True/False ??

Processing a range query using an unclustered index is always faster than using a file scan.

False

More Questions in GO Classes Test Series & Mock Tests.

Do you have CRYSTAL Clarity about All the types of Indexes now??

Tell me in Comments.

Index Structures

Index: A disk data structure
– enables efficient retrieval of a record
given the value (s) of certain attributes
– indexing attributes

Primary Index:
Index built on *ordering key* field of a file

Clustering Index:
Index built on *ordering non-key* field of a file

Secondary Index:
Index built on any *non-ordering* field of a file

Next Topic:

Ordered Indexes

Multi-Level