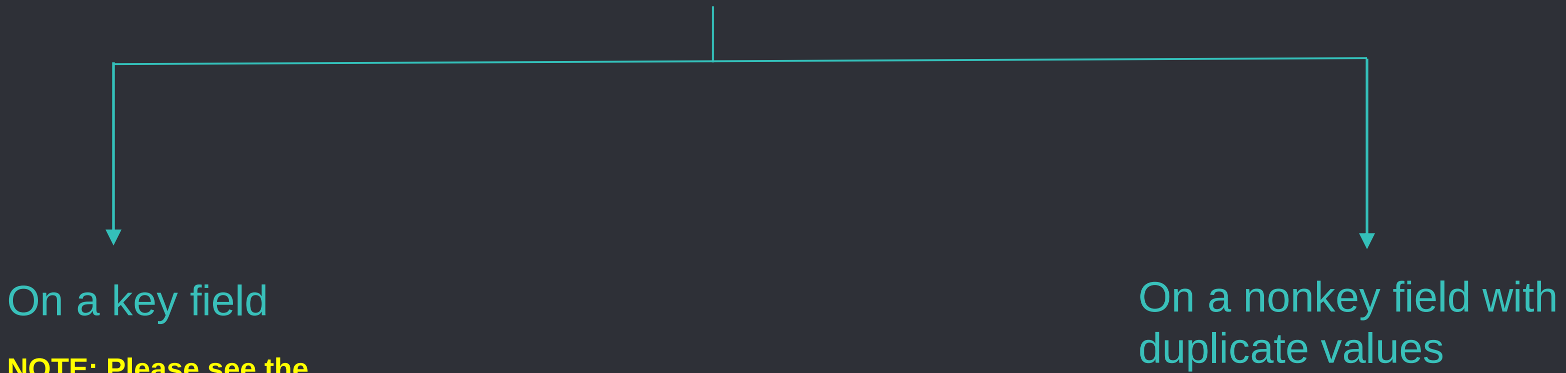
Chapter 18

Indexing Structures for Files

References/Resources:
R. Elmasri, S.B. Navathe, “Fundamentals of Database Systems”, 6th Edition, Pearson Education

Secondary Indexes



On a key field

On a nonkey field with duplicate values

NOTE: Please see the eg on slide no.4, then read the defination.

The secondary index (in both on a key and nonkey field) is 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.

Secondary Indexes on key field

- It is has distinct value for every record.
- It is a dense index.
- 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.
- We cannot use block anchor because data file is unordered.

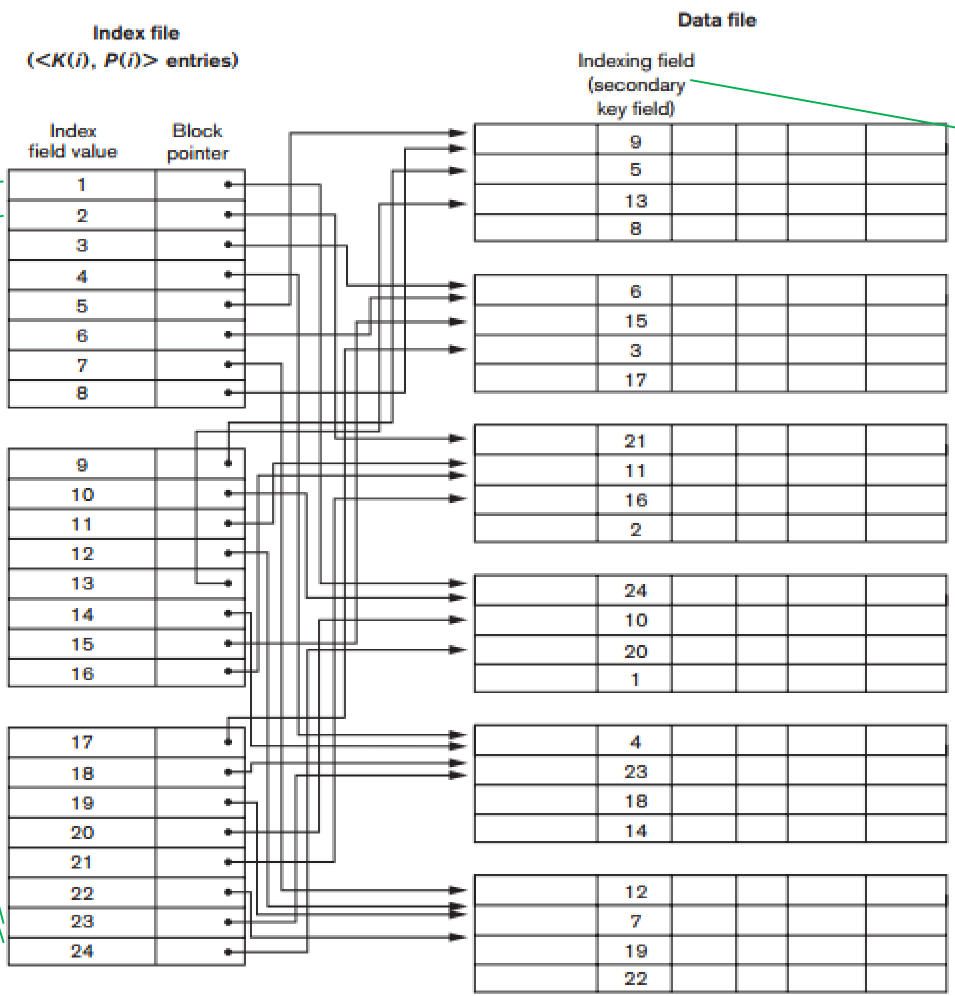
Disadvantages

- It is requires more storage space.
- The search time is long than a primary index, because of its larger no. of entries.

Please try numerical example 2 and 3

Block pointer not record pointer

Figure 18.4
A dense secondary index (with block pointers) on a nonordering key field of a file.



Index field value	Block pointer
1	●
2	●
3	●
4	●
5	●
6	●
7	●
8	●
9	●
10	●
11	●
12	●
13	●
14	●
15	●
16	●
17	●
18	●
19	●
20	●
21	●
22	●
23	●
24	●

	Indexing field (secondary key field)				
	9				
	5				
	13				
	8				

	6				
	15				
	3				
	17				

	21				
	11				
	16				
	2				

	24				
	10				
	20				
	1				

	4				
	23				
	18				
	14				

	12				
	7				
	19				
	22				

Values are unique,
∵ key field

NOTE:
Based on key value (hence unique) and the data file is unordered.

Secondary Index on Nonkey field

Option 1

Index File

1	
1	
2	
3	
3	

Second col duplicate index entry

Option 2

Index File

1	🟩 🟩
2	🟩
3	🟩 🟩 🟩

Second col a repeating field for pointers for each block

Option 3

Secondary Index on Nonkey field Option3 most commonly used

- A single entry for each index field value, and also create an extra level of indirection to handle the multiple pointers.
- 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.
- It is a nondense scheme.

Option 3

Contains all pointers of record 1

Contains all pointers of record 8

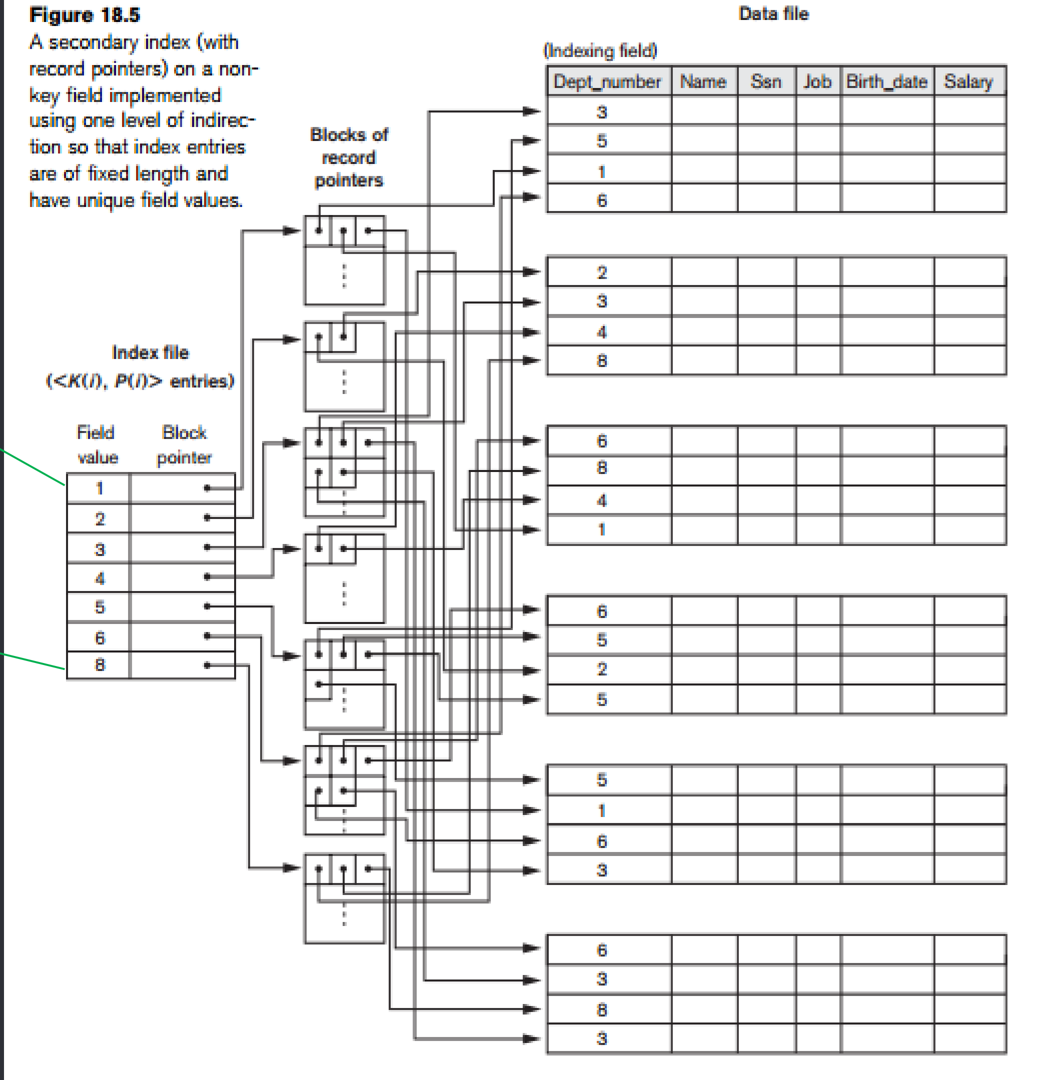


Figure 18.5
A secondary index (with record pointers) on a nonkey field implemented using one level of indirection so that index entries are of fixed length and have unique field values.

Index file
(<*K*(*i*), *P*(*i*)> entries)

Field value	Block pointer
1	
2	
3	
4	
5	
6	
8	

Data file

(Indexing field)

Dept_number	Name	Ssn	Job	Birth_date	Salary
3					
5					
1					
6					
2					
3					
4					
8					
6					
8					
4					
1					
6					
5					
2					
5					
5					
1					
6					
3					
6					
3					
8					
3					

NOTE: Based on nonkey value (hence duplicate) and the data file is unordered.

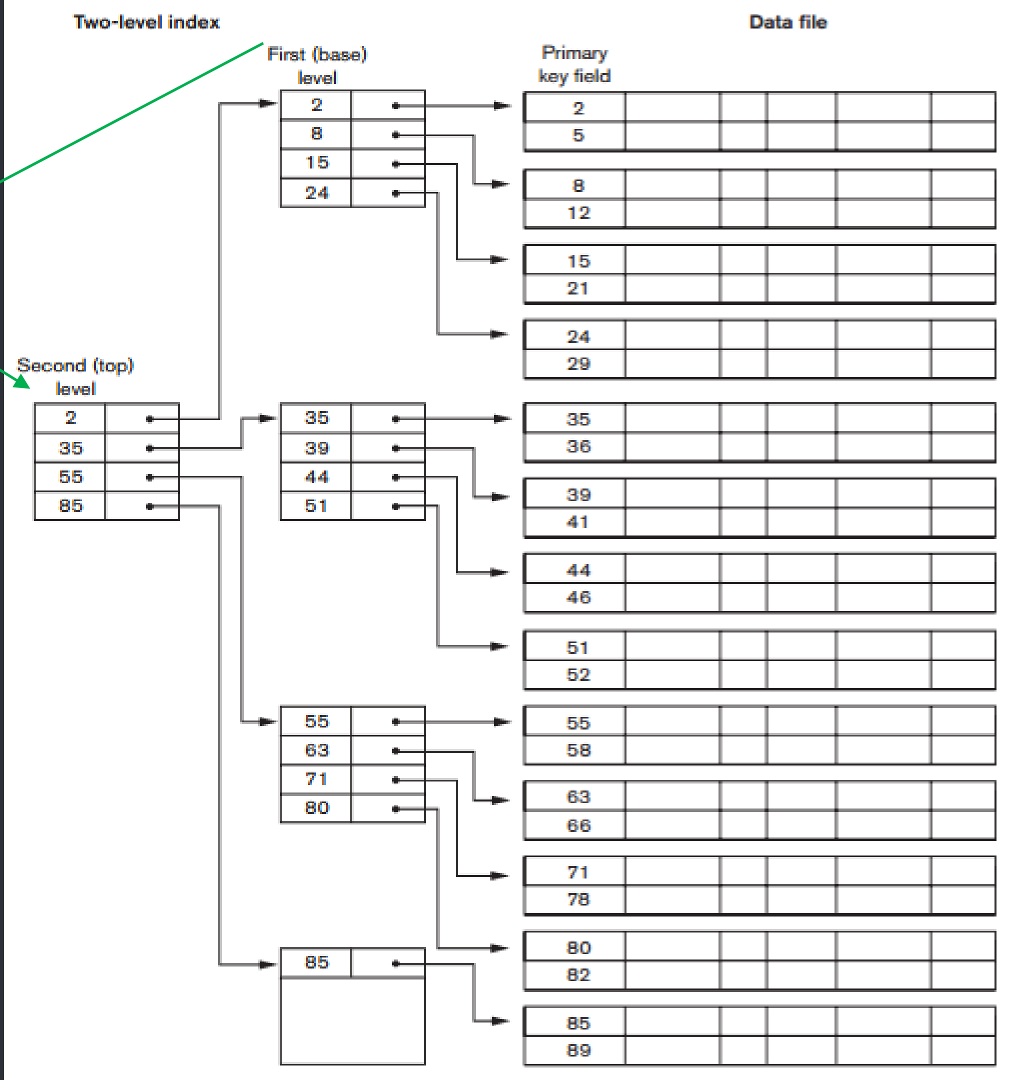
Advantage of Option3

Insertion and deletion of records in the data file is faster.

Disadvantage of Option3

Retrieval via the index requires one or more additional block accesses because of the extra level.

Multi level Index



Primary index of the first level

Multilevel Indexes

- It considers the index file, the first (or base) level of a multilevel index, as an ordered file with a distinct value for each K(i).
- Create a primary index for the first level; this index to the first level is called the second level of the multilevel index.
- The second level has one entry for each block of first level.
- The value bfr_i is called the fan-out of the multilevel index and it denoted by fo.
- All index entries are the same size with one field value and one pointer.
- If the first level has $\boldsymbol{r_i}$ entries and the blocking factor for the index is fo then, the no. of entries $\boldsymbol{r_2} = \left\lceil \frac{\boldsymbol{r_1}}{\boldsymbol{fo}} \right\rceil$ are needed at the second level of index.
- Similarly, third level entries $\boldsymbol{r_3} = \left\lceil \frac{\boldsymbol{r_2}}{\boldsymbol{fo}} \right\rceil$
- Multilevel scheme can be used in primary, clustering, or secondary—as long as the first-level index has distinct values for K(i) and fixed-length entries.

B Trees

Data records of data file are either in internal node or in leaf node.

B+ Trees

Data records of data file are exits in leaf node.

No need to study algorithms.

Go through the examples of section 18.3

Reference :-

R. Elmasri, S.B. Navathe, Fundamentals of Database systems (6^{th} edition), Pearson Education.

chapter - 16 _ Part I

Cover : A set of functional dependency F is said to cover another set of functional dependencies E if every FD in E is also in F^+, i.e, if every dependency in E can be inferred from F.
we can also say that E is covered by F.

Equivalence : Two set of functional ~~dependencie~~ dependencies E and F are equivalent if
$E^+ = F^+$.

∴ Equivalence means that every FD in E can be inferred from F and vice-versa.
i.e, E is equivalent to F, if both the conditions - E covers F & F covers E - hold.

Eg. For the following set of FD's . check whether $F \subseteq G$ or $G \subseteq F$ or $F = G$ or $F \neq G$.

$F = \{A \rightarrow C, AC \rightarrow D,$
$E \rightarrow AD, E \rightarrow H\}$

$G = \{A \rightarrow CD, E \rightarrow AH\}$

from ①

$F = A \rightarrow C$
$AC \rightarrow D$
$E \rightarrow AD$
$E \rightarrow H$

$\{A\}^+ = \{ACD\}$
$\{AC\}^+ = \{ACD\}$
$\{E\}^+ = \{EADHC\}$

$G = A \rightarrow CD$
$E \rightarrow AH$

$\underbrace{\{A\}^+ = \{ACD\} \quad \{E\}^+ = \{EADHC\}}$

↑

since, $F \subseteq G$ & $G \subseteq F$
$\therefore F = G$.

Q1. Find whether the two F.Ds (F & G) set are equivalent or not or whether one covers the other.

(i). $F : \{A \rightarrow B, B \rightarrow C, AB \rightarrow D\}$
$G : \{A \rightarrow B, B \rightarrow C, A \rightarrow C, A \rightarrow D\}$

(ii). $F : \{A \rightarrow B, B \rightarrow C, A \rightarrow C\}$
$G : \{A \rightarrow B, B \rightarrow C, A \rightarrow D\}$

(iii). $F = \{A \rightarrow B, B \rightarrow C, C \rightarrow A\}$
$G = \{C \rightarrow B, B \rightarrow A, A \rightarrow C\}$

~~Minimal cover: It is a set of functional dependencies E~~

Minimal cover: A minimal cover of a set of functional dependencies E is a minimal set of dependencies that is equivalent to E. we can always find at least one minimal cover F for any set of dependencies E.

Minimal cover means optimizing the set of F.D's; just like you can optimize the already written code.