



DATABASE SYSTEMS

CS – 355/CE – 373

Instructor: Maria N. Samad


November 27th, 2024

DENSE INDICES

- As data grows, the index file also grows, which means occupying more and more space
- If the indexing is not done on primary key, we can still rely on partial sequential search as in sparse indices – instead, in this case, the pointer will point to every search key's first instance. This can help reduce the space overhead to a certain extent
- Example from the class activity on Ordered Indices, we got dense clustered index on staffNo as:

Search Key	Pointer	staffNo	dentistName	patNo	patName	appointment date time	surgeryNo
S1011		S1011	Tony Smith	P100	Gillian White	12-Sep-13 10.00	S15
S1024		S1011	Tony Smith	P105	Jill Bell	12-Sep-13 12.00	S15
S1032		S1024	Helen Pearson	P108	Ian MacKay	12-Sep-13 10.00	S10
		S1024	Helen Pearson	P108	Ian MacKay	14-Sep-13 14.00	S10
		S1032	Robin Plevin	P105	Jill Bell	14-Sep-13 16.30	S15
		S1032	Robin Plevin	P110	John Walker	15-Sep-13 18.00	S13

Dense Clustered Index File



DENSE INDICES

- We can make use of the idea of sparse indices and use the following technique instead:

Search Key	Pointer	staffNo	dentistName	patNo	patName	appointment date time	surgeryNo
S1011	→	S1011	Tony Smith	P100	Gillian White	12-Sep-13 10.00	S15
S1024	→	S1011	Tony Smith	P105	Jill Bell	12-Sep-13 12.00	S15
S1032	→	S1024	Helen Pearson	P108	Ian MacKay	12-Sep-13 10.00	S10
	→	S1024	Helen Pearson	P108	Ian MacKay	14-Sep-13 14.00	S10
	→	S1032	Robin Plevin	P105	Jill Bell	14-Sep-13 16.30	S15
	→	S1032	Robin Plevin	P110	John Walker	15-Sep-13 18.00	S13

Dense Clustered Index File

- This reduces the space because now you won't have two entries for references or pointers
- However, this will still be a problem if this dense indexing was being done on single primary keys, then we will need as many entries in the index file as there are primary keys in the table, so what to do then?
 - Use Multilevel Indexing

MULTILEVEL INDICES

- If index does not fit in memory, access becomes expensive.
- Solution: treat index kept on disk 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 in main memory, yet another level of index can be created, and so on.

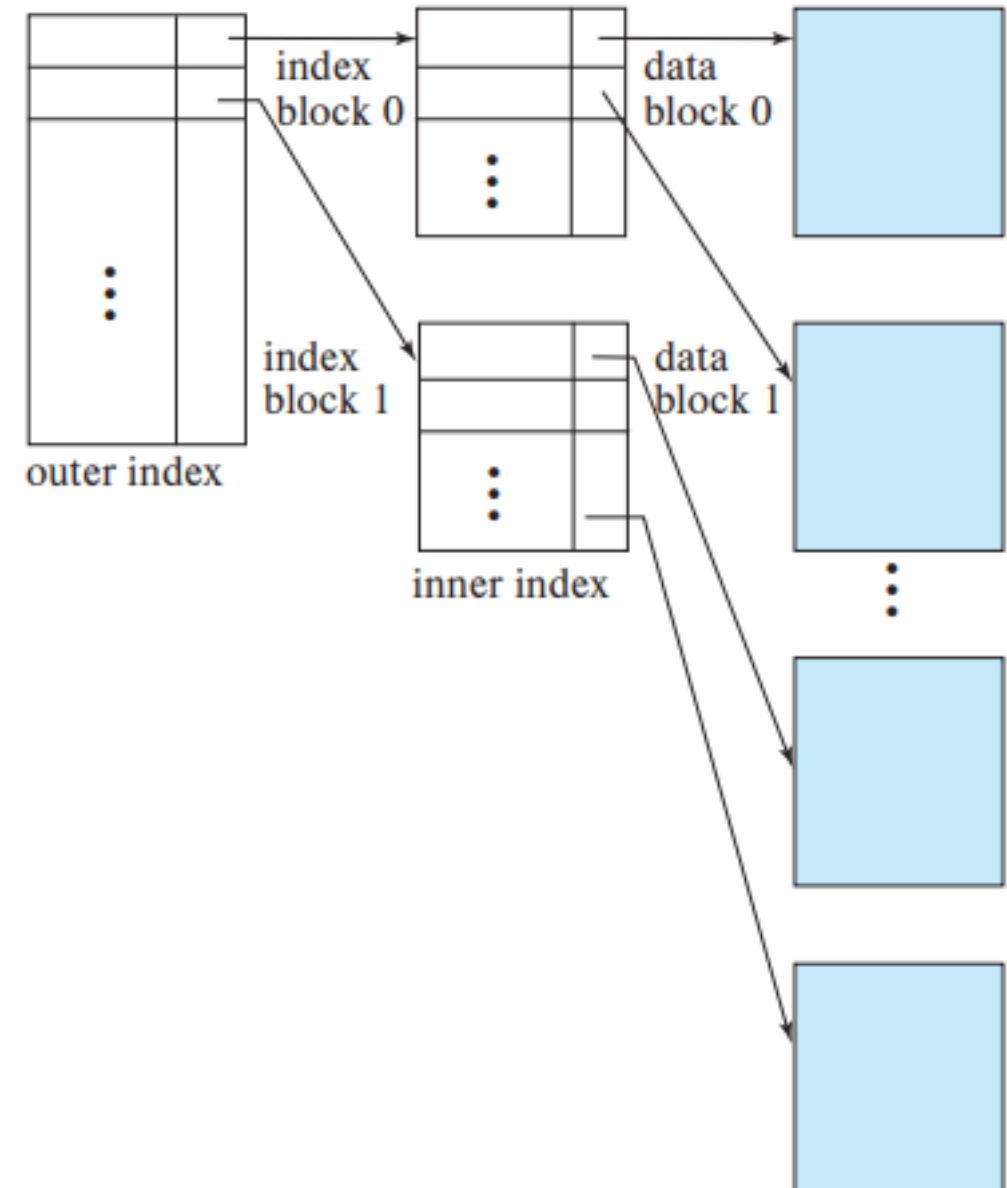


Figure 14.5 Two-level sparse index.

MULTILEVEL INDICES

- Example:
 - If a relation has 1,000,000 records, and in case of dense indexing on primary key, we will need 1 million entries in the index file
 - To store these 1 million entries of index structure, it will require some space as well.
 - This will be smaller space as compared to the relation itself, but some amount of space nonetheless
 - Assume we can store first 100 index entries in a 4KB block of secondary memory, which means we will need 10,000 blocks of secondary storage to store the index file completely

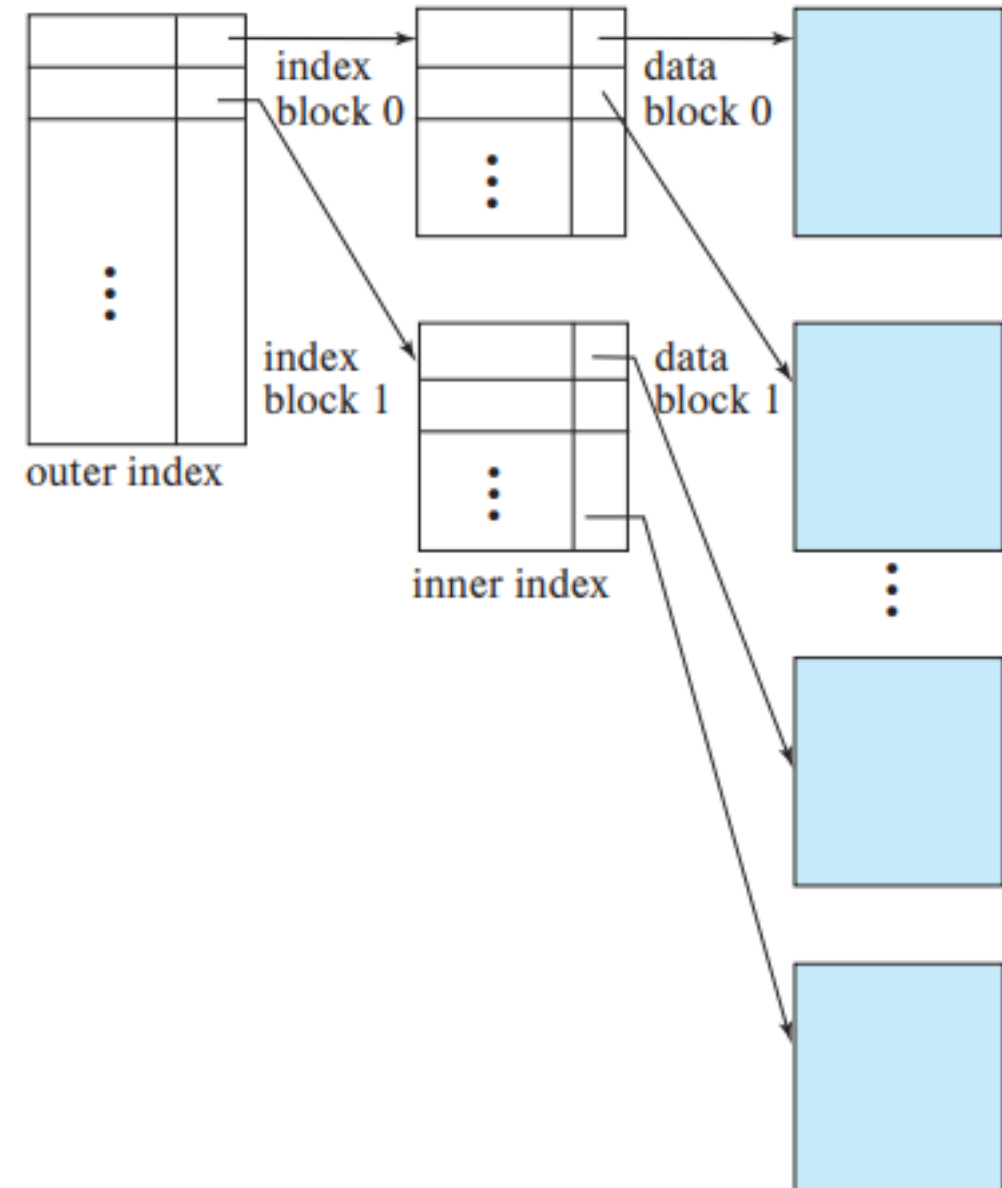


Figure 14.5 Two-level sparse index.

MULTILEVEL INDICES

- Example:
 - The amount of space may not be an issue, but still need to worry about how to access it?
 - In order to access the index file, it should be brought from the secondary storage to the main memory
 - If the entire index file can be brought in the main memory then there's no problem, as MM is direct access and will result in faster results too
 - However, what if we are unable to bring the entire index file into the MM at once, which means the system will need to first look for the block that contains the required index in the secondary storage and then bring in that particular block in the MM

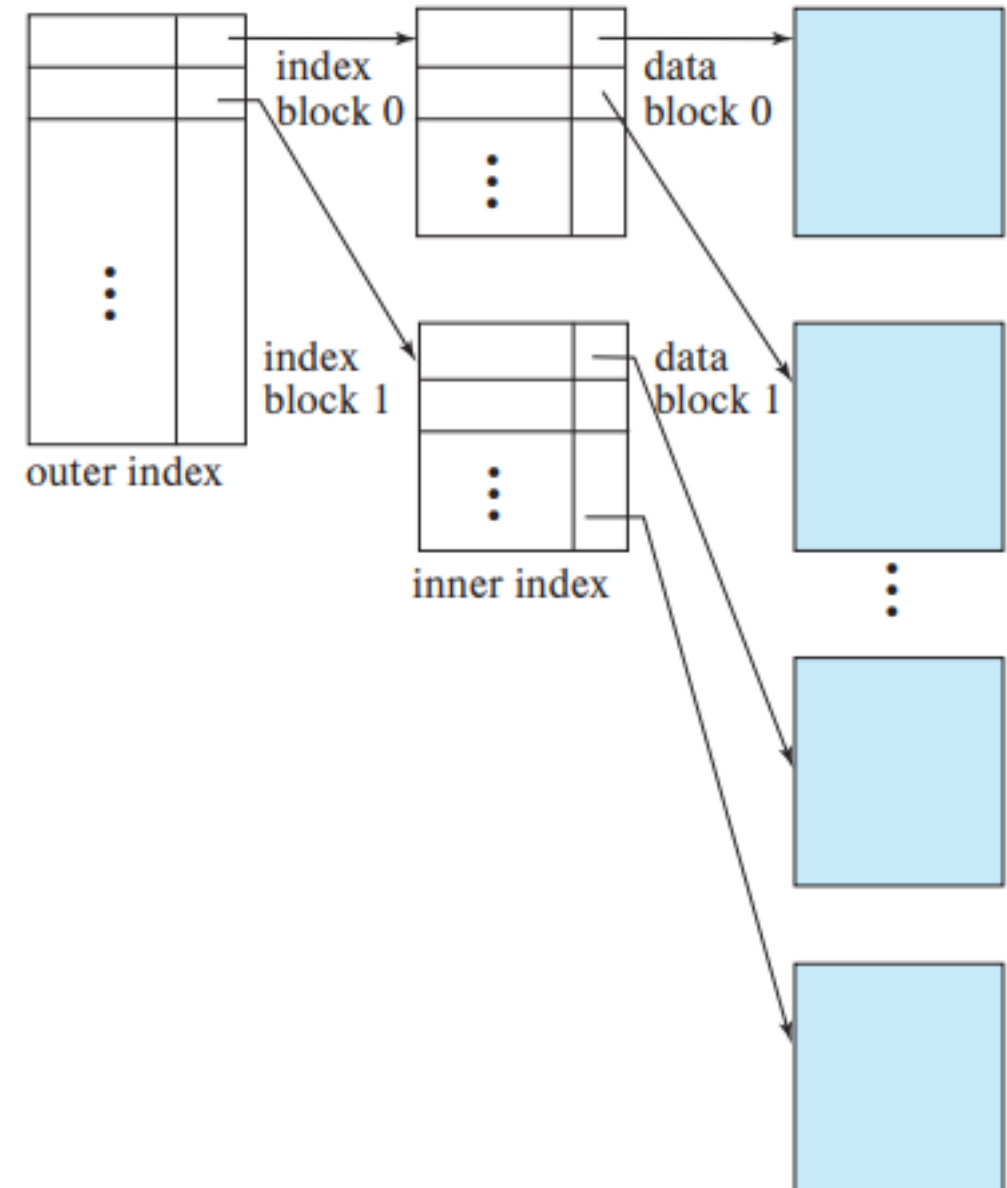


Figure 14.5 Two-level sparse index.

MULTILEVEL INDICES

- Example:
 - This is where multilevel indexing becomes useful in giving efficient results
 - Indexing is applied to the index file, which is an ordered, sequential file; and then we can use sparse indexing to utilize some space as well.
 - This means the outer index will be sparse, i.e. containing some references of the actual indices
 - This can become faster to process because now the outer index file is much smaller in size which may be brought in the main memory completely.

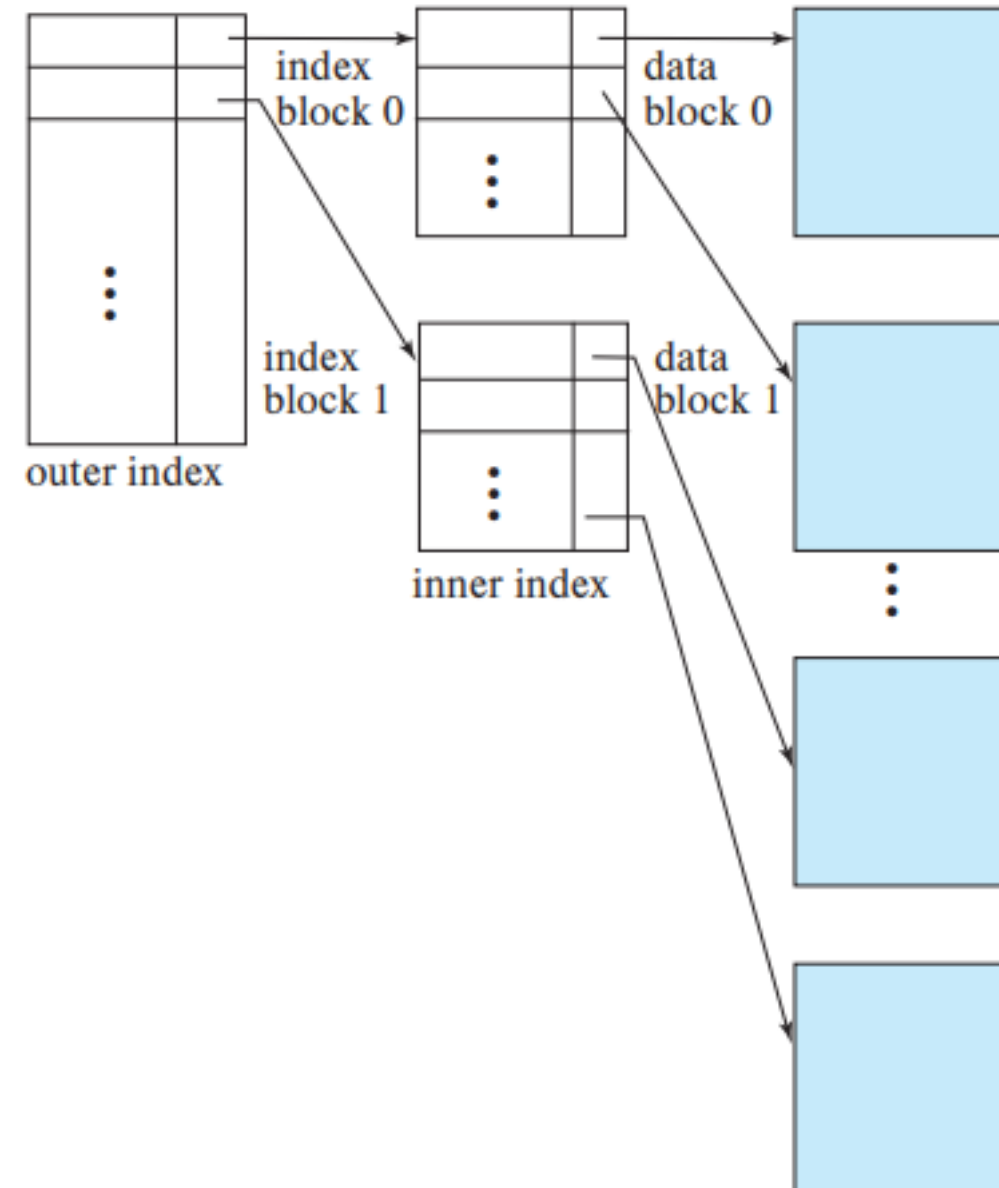


Figure 14.5 Two-level sparse index.

MULTILEVEL INDICES

- Example:
 - Searching for the outer index will be as fast as the MM processes, and then the desired block in secondary storage can be reached directly, instead of searching for the entire structure in the secondary storage
 - After which the process is as fast as any indexing of a small sized data set works.

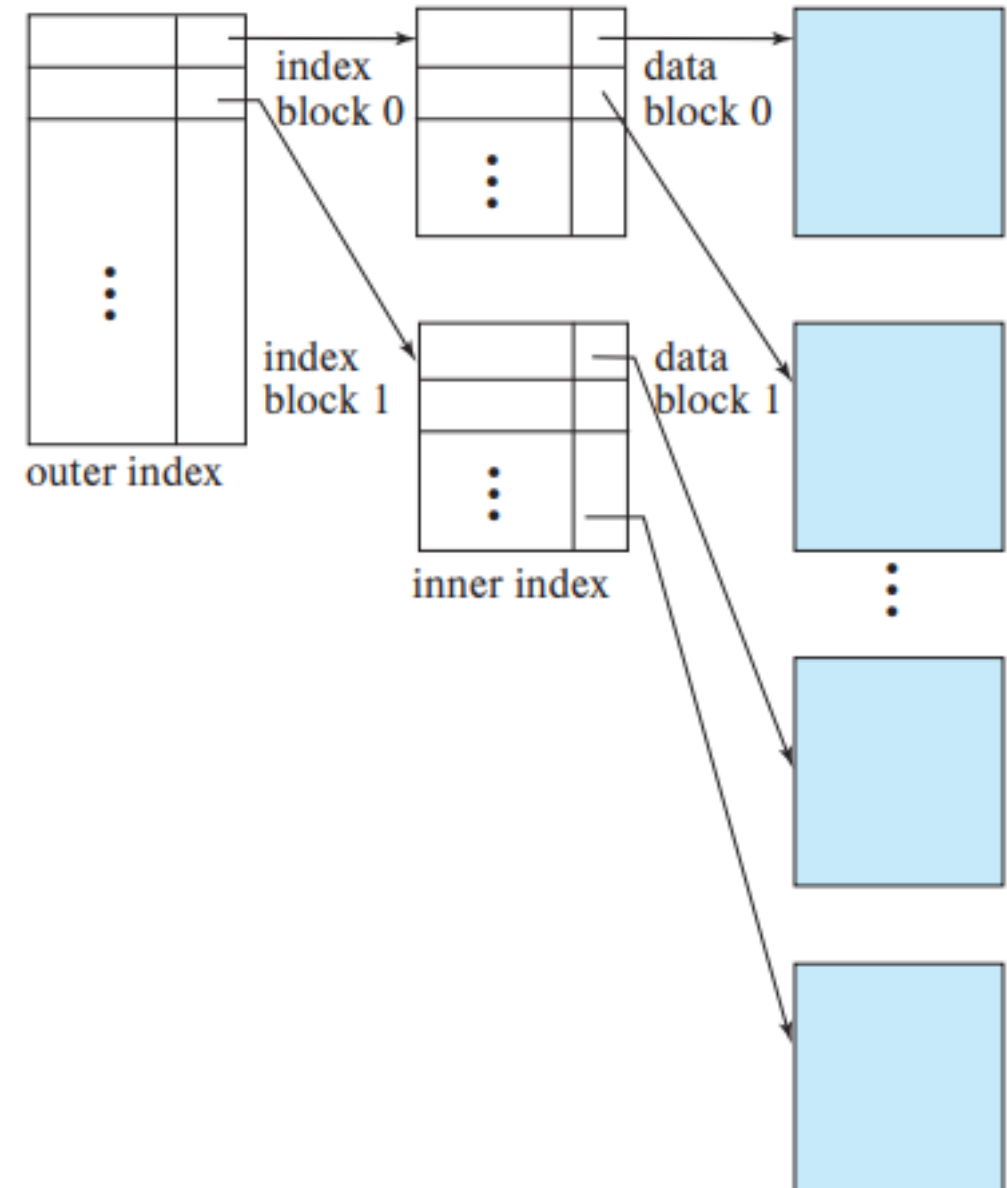


Figure 14.5 Two-level sparse index.

MULTILEVEL INDICES

- Example:
 - If the outer index structure is still too big to fit in the main memory, it can be further indexed to get three levels of indexing
 - Having more levels will be slower than having just two levels, but it will still be faster than searching at one level.
 - The number of levels to be added depends on the size of data, and the time it requires to go through these levels.
 - However, there is no restriction on how many levels the system can have

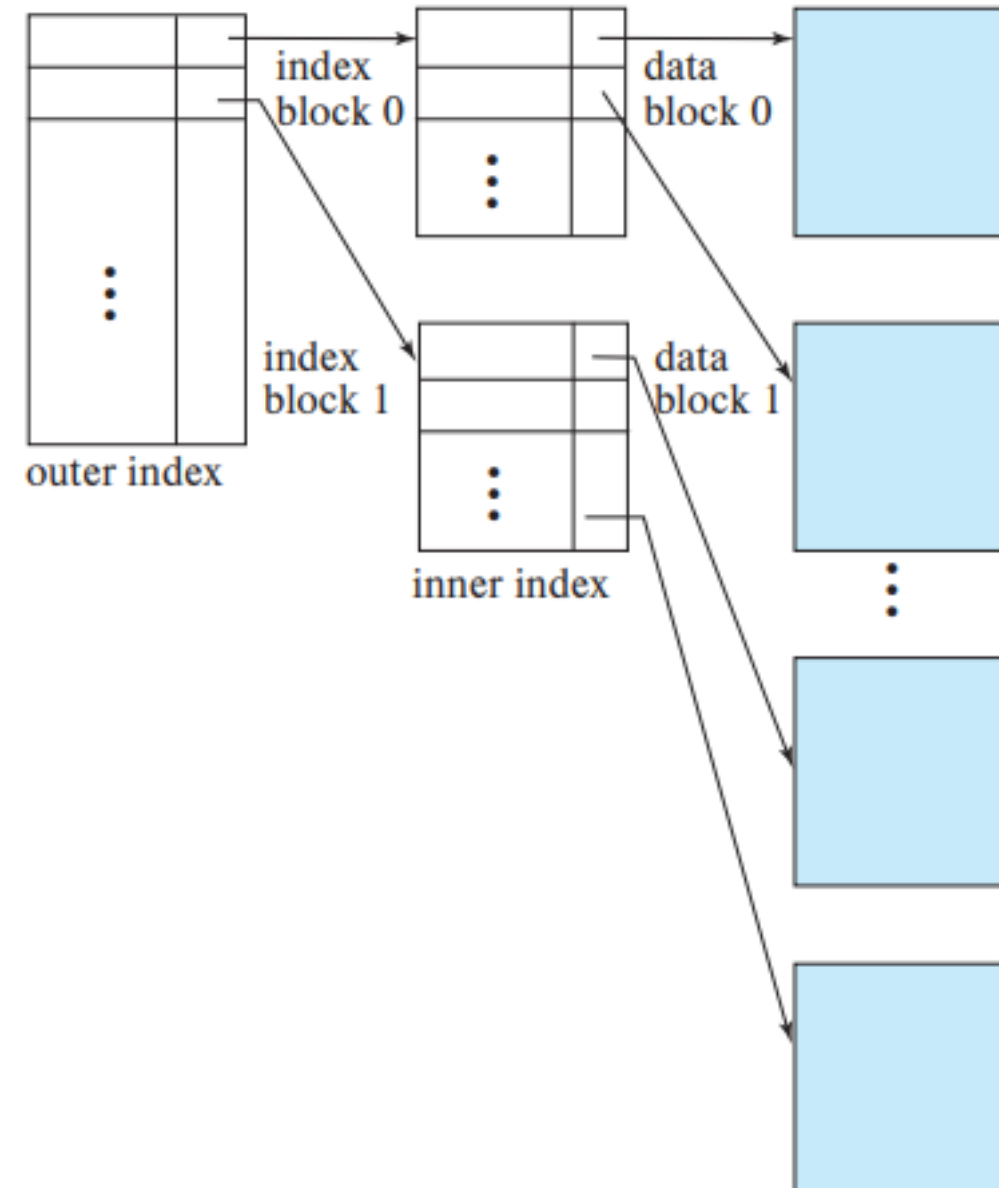


Figure 14.5 Two-level sparse index.

MULTILEVEL INDICES

- Example:
 - For a library management system, there are 100 entries in a relation for books
 - Assume indexing is done on BookID, which is a Primary key of the table and is generated by auto-incrementing it each time, starting from 1.
 - Answer the following questions:
 1. What kind of indexing will be used? Give reason for your answer.
 - a. Clustered or Non clustered?
 - b. Dense or Sparse?
 2. Assume a single block in Main Memory (MM) can hold 10 entries at a time. How many memory accesses would we need to access the search key value of 95. Assume Dense indexing irrespective of your choice in part 1
 3. Based on your answer to part 2, briefly explain if multi-level indexing is needed or not? Give reason for your choice.

MULTILEVEL INDICES

- Example:
 - For a library management system, there are 100 entries in a relation for books
 - Assume indexing is done on BookID, which is a Primary key of the table and is generated by auto-incrementing it each time, starting from 1.
 - Answer the following questions:
 4. Apply multi-level indexing until optimized, keeping the MM block size, as specified in previous parts. For each level, answer the following:
 - i. Calculate the number of entries for each index file
 - ii. Total number of memory accesses required
 5. What are the steps to access record with BookID = 95 after applying multi-level indexing?
 6. How many memory accesses are required in single-level indexing? Multi-level indexing? Assume accessing the record in the table as one memory access to answer the question

MULTILEVEL INDICES

- Solution:

1. What kind of indexing will be used?

- a. Clustered or Non clustered?

- As BookID is a primary key, we will need as many entries as there are records in the table, i.e. 100
- The table itself will have sorted BookIDs because of auto-increment feature, the index will be considered as Clustered

- b. Dense or Sparse?

- Both Dense and Sparse indexing can be used here as it is a Clustered index
- As there are only 100 entries, it won't be too much of a space overhead so we can go with Dense clustered indexing
- If there were thousands of entries instead of just 100, then we can make a sparse index to reduce the space overhead

MULTILEVEL INDICES

- Solution:

2. Assume a single block in Main Memory (MM) can hold 10 entries at a time. How many accesses would we need to access the search key value of 95. Assume Dense indexing irrespective of your choice in part 1

- $$\text{Total Number of accesses} = \frac{\text{Total Number of entries in the index file}}{\text{Number of entries per MM block}}$$

- $$\text{Total Number of accesses} = \frac{100}{10} = 10 \text{ accesses}$$

3. Based on your answer to part 2, briefly explain if multi-level indexing is needed or not? Give reason for your choice?

- The number of accesses is 10, which may not seem a bigger number, but as the number of records increase, this will make the search queries slower for the user, so it is better to implement multi-level indexing, so that even when the number of records increase, it will still give output efficiently

MULTILEVEL INDICES

- Solution:

4. Apply multi-level indexing until optimized, keeping the MM block size, as specified in previous parts. For each level, answer the following:

- i. Calculate the number of entries for each index file
- ii. Total number of memory accesses required

Ans. First introduce another level, i.e. 2nd level indexing with sparse indexing, and answer the questions to deduce if this is optimized at this level or not

i. Number of entries at 2nd level = $\frac{100}{10} = 10$

ii. Total number of memory accesses required = $\frac{10}{10} = 1$

- This means we can bring the entire 2nd level index file in the MM which will give us the reference to the desired record entry in 1st level making it efficient

MULTILEVEL INDICES

- Solution:
 5. What are the steps to access record with BookID = 95 after applying multi-level indexing?
 - First it accesses the highest level index file, which in this case is 2nd level, and look for search key 95.
 - It won't find 95 directly in it, so it looks for the closest smaller value to 95, which in this case is going to be 91. You don't have to calculate the actual values to answer questions like these because we can be dealing with huge number of records. The values are being referenced here for explanation purposes only.
 - From that entry (in this case 91), the pointer field will give the address of 91 in the 1st level index file, and access that entry directly.
 - In the 1st level index file, it will then access the next 10 entries (because MM block size = 10)
 - MM then finds 95 entry directly, and from that entry, it reads the pointer address of this search key in the original table
 - Finally the record is accessed

MULTILEVEL INDICES

- Solution:
 6. How many memory accesses are required in single-level indexing? Multi-level indexing? Assume accessing the record in the table as one memory access to answer the question
 - Single-level indexing:
 - Number of accesses to get the entry for search key = 10
 - Fetching record from the original table = 1
 - Total number of accesses = 11

MULTILEVEL INDICES

- Solution:

6. How many memory accesses are required in single-level indexing? Multi-level indexing? Assume accessing the record in the table as one memory access to answer the question

- Multi-level indexing:

- Fetching 2nd level index file = 1
- Fetching desired block of 1st level index file = 1
- Fetching record from the table = 1
- Total number of accesses = 3

MULTILEVEL INDICES

- Activity Sheet

MULTILEVEL INDICES

- Activity Sheet Solution:
 - [Single vs Multi-level Indices Solution](#)