11.15 When is it preferable to use a dense index rather than a sparse index? Explain your answer.

Answer: It is preferable to use a dense index instead of a sparse index when the file is not sorted on the indexed field (such as when the index is a secondary index) or when the index file is small compared to the size of memory.

11.16 What is the difference between a clustering index and a secondary index?

Answer: The clustering index is on the field which specifies the sequential order of the file. There can be only one clustering index while there can be many secondary indices.

11.22 Suppose there is a relation $r(A, B, C)$, with a B$^+$-tree index with search key (A, B).

a. What is the worst-case cost of finding records satisfying $10 < A < 50$ using this index, in terms of the number of records retrieved n_1 and the height h of the tree?

b. What is the worst-case cost of finding records satisfying $10 < A < 50 \wedge 5 < B < 10$ using this index, in terms of the number of records n_2 that satisfy this selection, as well as n_1 and h defined above?

c. Under what conditions on n_1 and n_2 would the index be an efficient way of finding records satisfying $10 < A < 50 \wedge 5 < B < 10$?

Answer:

a. What is the worst case cost of finding records satisfying $10 < A < 50$ using this index, in terms of the number of records retrieved n_1 and the height h of the tree?
This query does not correspond to a range query on the search key as the condition on the first attribute if the search key is a comparison condition. It looks up records which have the value of A between 10 and 50. However, each record is likely to be in a different block, because of the ordering of records in the file, leading to many I/O operation. In the worst case, for each record, it needs to traverse the whole tree (cost is h), so the total cost is $n_1 * h$.

b. What is the worst case cost of finding records satisfying $10 < A < 50 \wedge 5 < B < 10$ using this index, in terms of the number of records n_2 that satisfy this selection, as well as n_1 and h defined above.
This query can be answered by using an ordered index on the search key (A, B). For each value of A this is between 10 and 50, the system located records with B value between 5 and 10. However, each record could is likely to be in a different disk block. This amounts to executing the query based on the condition on A, this costs $n_1 * h$. Then these records are checked to see if the condition on B is satisfied. So, the total cost in the worst case is $n_1 * h$.

c. Under what conditions on n_1 and n_2 would the index be an efficient way of finding records satisfying $10 < A < 50 \wedge 5 < B < 10$.
n_1 records satisfy the first condition and n_2 records satisfy the second condition. When both the conditions are queried, n_1 records are output in the first stage. So, in the case where $n_1 = n_2$, no extra records are output in the furst stage. Otherwise, the records which dont satisfy the second condition are also output with an additional cost of h each (worst case).