Solutions

Answer 11.1 The answer to each question is given below. Note that the unclustered index (on <sal>) has 10000 * 1/5 = 2000 leaf pages, and a height of $\log_{100}2000 = 2$ (as the index fanout - the number of index entries per page - is 100) and clustered (<*age, sal*>) has 10000 * 2/5 = 4000 leaf pages, and a height of $\log_{50}4000 = 3$ (as the index fanout is 50).

1.

(a) *sal* > 100 For this condition, a filescan is the best option, since a clustered index does not exist on *sal*. Using the unclustered index would accrue a cost of $2(lookup) + 10,000\frac{pages*20bytes}{100bytes} * 0.1$ for the B+ index scan plus $10,000pages * 20tuplesperpage * 0.1$ for the lookup = 20202, and would be inferior to the filescan cost of 10000.

(b) *age* = 25 The clustered B+ tree index would be the best option here, with a cost of $3(lookup) + 10000pages * 0.1(selectivity) = 1003$. Although the hash index has a lesser lookup time (roughly 1.2), the potential number of record lookups $(10000pages * 0.1 * 20tuplesperpage = 20000)$ renders the clustered index more efficient.

(c) *age* > 20 Again the clustered B+ tree index is the best of the options presented; the cost of this is $3(lookup) + 10000pages * 0.1(selectivity) = 1003$.

(d) *eid* = 1000 Since *eid* is a candidate key, one can assume that only one record will be in each bucket. Thus, the total cost is roughly 1.2 (lookup) + 1 (record access) which is 2 or 3.

(e) *sal* > 200 ∧ *age* > 30. This case is similar to the age > 20 one when we first evaluate the age > 20 clause and the cost is: $3(lookup) + 10000pages * 0.1(selectivity) = 1003$.

(f) *sal* > 200 ∧ *age* = 20. Similar to the previous part, except now we do not need to scan all matching index pages for age. We lookup for the first data page with age = 20, and start scanning all data pages for sal > 200 until we reach age 21. Total cost: $3(lookup) + 10000pages * 0.1(selectivity) * 0.1(selectivity) = 103$.

(g) *sal* > 200 ∧ *title* = '*CFO*' In this case, the filescan is the best available method to use, with a cost of 10000.

(h) *sal* > 200 ∧ *age* > 30 ∧ *title* = '*CFO*' Here an age condition is present, so the clustered B+ tree index on < *age,sal* > can be used. Here, the cost is $3(lookup) + 10000pages * 0.1(selectivity) = 1003$.

2.

(a) *sal* > 100 Since the desired result is only the average salary, an index-only scan can be performed using the unclustered B+ tree on *sal* for a cost of $2(lookup) + 10000 * 0.1 * 0.2(smallerindextuples) = 202.$

(b) *age* = 25 For this case, the best option is to use the clustered index on < *age,sal* > and thus perform index-only scan. The cost of this operation is $3(lookup) + (10000 * 0.4) * 0.1 = 403.$

(c) *age* > 20 Similar to the *age* = 25 case, this will cost 403 using the clustered index.

(d) *eid* = 1000 Being a candidate key, only one relation matching this should exist. Thus, using the hash index again is the best option, for a cost of 1.2 (hash lookup) + 1 (relation retrieval) = 2.2.

(e) *sal* > 200 ∧ *age* > 30 Using the clustered B+ tree again as above is the best option, with a cost of 403.

(f) *sal* > 200 ∧ *age* = 20 Similarly to the *sal* > 200 ∧ *age* = 20 case in the previous problem, this selection should use the clustered B+ index for an index only scan, costing $3(lookup) + 4000 * 0.1(selectivityforage) * 0.1(selectivityforsal) = 43.$

(g) *sal* > 200∧*title* = '*CFO*' In this case, an index-only scan may not be used, and individual relations must be retrieved from the data pages. The cheapest method available is a simple filescan, with a cost of 10000 I/Os.

(h) *sal* > 200 ∧ *age* > 30 ∧ *title* = '*CFO*' Since this query includes an age restriction, the clustered B+ index over < *age,sal* > can be used; however, the inclusion of the title field precludes an index-only query. Thus, the cost will be: $3(lookup) + 10000pages * 0.1(selectivity) = 1003.$

3.

(a) *sal* > 100 The best method in terms of I/O cost requires usage of the clustered B+ index over < *age,sal* > in an index-only scan. Also, this assumes the ability to keep a running average for each age category. The total cost of this plan is: $10000 * 0.4(index - onlyscan) = 4000.$ Note that although *sal* is part of the key, since it is not a *prefix* of the key, the entire list of pages must be scanned.

(b) *age* = 25 Again, the best method is to use the clustered B+ index in an index-only scan. For this selection condition, this will cost $3(agelookup) + 4000pages * 0.1(selectivityonage) = 403.$

(c) *age* > 20 This selection uses the same method as the previous condition, the clustered B+ tree index over < *age,sal* > in an index-only scan, for a total cost of 403.

(d) *eid* = 1000 As in previous questions, *eid* is a candidate field, and as such should have only one match for each equality condition. Thus, the hash index over *eid* should be the most cost effective method for selecting over this condition, costing 1.2 (hash lookup) + 1 (tuple retrieval) = 2.2.

(e) *sal* > 200 ∧ *age* > 30 This can be done with the clustered B+ index and an index-only scan over the < *age,sal* > fields. The total estimated cost is $3(lookup) + 4000pages * 0.1(selectivityonage) = 403.$

(f) *sal* > 200 ∧ *age* = 20 This is similar to the previous selection conditions, but even cheaper. Using the same index-only scan as before (the clustered B+ index over < *age,sal* >), the cost should be $3 + 10000 * 0.4 * 0.1(ageselectivity) * 0.1(salselectivity) = 43.$

(g) *sal* > 200∧ *title* = '*CFO*' Since we have a clustered < *age,sal* > index, we can use the ordering of the data to answer this query with a simple filescan while keeping running average for each age and each qualifying tuple.

(h) *sal* > 200 ∧ *age* > 30 ∧ *title* = '*CFO*' Using the clustered B+ tree over < *age,sal* > would accrue a cost of $3 + 10000 * 0.1(selectivityofage) = 1003.$

4.

(a) *sal* > 100 Since we only need sal and age attributes, the best operation involves scanning the clustered index on <age, sal>, filtering on sal>100 and sorting the result. We can do initial sorting pass using 10 buffers (because 1 is needed for scanning the index), and the merging passes using 11 buffers (1 is used for output, so the base of the log in the formula will be 10). Given the selectivity of the predicate result has 400 pages that can be sorted in $log_{10}\frac{400}{10} = 2$ passes. Finally, we do not need to write out the final result and we can do the aggregation during the last pass. The final cost is: $4000(readingtheindex) + 400(firstpass) + 400 * 2 * 2(sorting) - 400(writingouttheresult) = 5600$.

(b) *age* = 25 This case is similar to the previous, only we can probe the clustered index and then read the 400 qualifying pages.

Option 1: sort on sal the qualifying pages, where the cost is:$3(lookup) + 400(readingtheindex) + 400(firstpass) + 400 * 2 * 2(sorting) - 400(writingouttheresult) = 2003$.

Option 2: Since the predicate states that age has value of 25, we could skip sorting, where the cost is: $3(lookup) + 400(readingtheindex).$

(c) *age* > 20 Using the same approach as the previous case, the cost is 2003.

(d) *eid* = 1000 Being a candidate key, only one relation should match with a given *eid* value. Thus, the estimated cost should be 1.2 (hash lookup) + 1 (tuple retrieval).

(e) *sal* > 200∧ *age* > 30 In this case, we can also use the clustered index on <age, sal> similarly to the case b). We first probe to find first tuple satisfying age > 30, then scan the 400 qualifying pages and filter on the predicate sal > 200. The qualifying tuples can be stored in 40 pages ($4000 * 0.1(ageselectivity) * 0.1(salselectivity)$). Using other 10 buffers we can produce 4 sorted runs of 10 pages, which we can merge in one pass during which we also perform aggregation. The total cost is: $3(lookup) + 400(readingtheindex) + 40(firstpass) + 40(merging) = 483.$

(f) *sal* > 200 ∧ *age* = 20 Similarly to 3f) we probe the clustered index on <age,sal> and read the qualifying tuples from 40 pages of the index.
Option 1: sort on sal the qualifying page, where the cost is: $3(lookup) + 400(readingtheindex) + 40(firstpass) + 40(merging).$
Option 2: Since the predicate states that age has value of 20, we could skip sorting, where the cost is: $3(lookup) + 40(readingtheindex) = 43.$

(g) *sal* > 200 ∧ *title* = '*CFO*' In this case, we need to do a filescan, perform the filters to find $10000 * 20 * 0.1(selectivityforsal) * 0.1(selectivityfortitle) = 2000$ and project sal and age attributes. These attributes require 40 pages. We can use 10 buffers to generate the sorter runs that we can then merge and perform aggregation in another pass for the total cost of: $10000(filescan) + 40(firstpass) + 40(merging) = 10080.$ If we tried to use index on sal, we would need to scan all index pages that satisfy sal > 200 and retrieve all matching tuples for the cost of $2(lookup) + 2000 * 0.1(indexscan) + 10000 * 20 * 0.1(selectivityforsal) = 20202$. There are no additional costs because aggregation can be done by keeping running averages while retrieving tuples.

(h) *sal* > 200 ∧ *age* > 30 ∧ *title* = '*CFO*' In this case, the number of tuples that satisfy all 3 conditions is $10000 * 20 * 0.1(selectivityforsal) * 0.1(selectivityforage) * 0.1(selectivityfortitle) = 200$ We need two attributes, sal and age, sized 20 bytes each, to compute the result and we can store 200 pairs of them on 4 buffer pages. This means that contrary to previous cases, we can do the aggregation in memory (without external sort). We can retrieve the matching values by probing the clustered index and scanning the data pages for the cost of $3(lookup) + 10000 * 0.1(selectivityofage) = 1003$

5.
(a) *sal* > 200∨ *age* = 20 In this case, a filescan would be the most cost effective, because the most cost effective method for satisfying *sal* > 200 alone is a filescan.

(b) *sal* > 200 ∨ *title* = '*CFO*' Again a filescan is the better alternative here, since no index at all exists for *title*.

(c) *title* = '*CFO*' ∧ *ename* = '*Joe*' Even though this condition is a conjunction, the filescan is still the best method, since no indexes exist on either *title* or *ename*.

Answer 11.2 The answer to each question is given below.
1. E.did, D.did
2. E.sal, E.did, D.did
3. E.sal, E.did, D.did, D.floor
4. E.did, D.did
5. D.floor, D.budget
6. D.floor, D.budget

Answer 11.3 Note that the number of leaf pages of indexes are 10000 *1/4 = 2500, 10000 * 2/4 = 5000 and 10000 * 3/4 = 7500 respectively for indexes on 1, 2 and 3 fields. With a fanout of 50, the height and therefore, the cost of lookup for these indexes are ceil($\log_{50}$2500) = 2, ceil($\log_{50}$5000) = 3 and ceil($\log_{50}$7500) = 3

1. (a) The best plan, a B+ tree search, would involve using the B+ tree to find the first *title* index such that *title*='CFO', cost = 2. Then, due to the clustering of the index, the relation pages can be scanned from that index's reference $cost = 10000 * 10\% + 2(lookup) = 1002(totalcost)$.
The cost for a tree lookup is equal to the number of index pages need to be read. An index page can contain 50 keys. Since we want to index 2500 pages, we will have 2500 leaf pages. As such, the height of the tree will be $\log_{50}$2500 = 2. Hence, a lookup requires fetching 2 index pages before reaching the leaf level.

(b) An unclustered index would preclude the low cost of the previous plan and necessitate the choice of a simple filescan, cost = 10000, as the best.

(c) Due to the WHERE clause, the clustered B+ index on ename doesn't help at all. The best alternative is to use a filescan, cost = 10000.

(d) Again, as in the previous answer, the best choice is a filescan, cost = 10000.

(e) Although the order of the B+ index key makes the tree much less useful, the leaves can still be scanned in an index-only scan, and the increased number of tuples per page lowers the I/O cost. Cost = 10000 * .5 = 5000.

2.
(a) A clustered index on *title* would allow scanning of only the 10% of the desired tuples. Thus the total cost is $2(lookup) + 10000 * 10\% = 1002$.

(b) A clustered index on *dname* works functionally in the same manner as that in the previous question, for a cost of 1002. The *ename* field still must be retrieved from the relation data pages.

(c) In this case, using the index lowers the cost of the query slightly, due to the greater selectivity of the combined query and to the search key taking advantage of it. The total cost = $3(lookup) + 10000 * 5\% = 503$.

(d) Although this index does contain the output field, the *dname* still must be retrieved from the relational data pages, for a cost of $3(lookup) + 10000 * 10\% = 1003$.

(e) Since this index contains all three indexes needed for an index-only scan, the cost drops to $3(lookup) + 10000 * 5\% *.75$.

(f) Finally, in this case, the prefix cannot be matched with the equality information in the WHERE clause, and thus a scan would be the superior method of retrieval. However, as the clustered B+ tree's index contains all the indexes needed for the query and has a smaller tuple, scanning the leaves of the B+ tree is the best plan, costing $10000 *.75 = 7500$ I/Os.

3.
(a) Since *title* is the only attribute required, an index-only scan could be performed, with a running counter. This would cost $10000 *.25(index - onlyscan, smallertuples) = 2500$.

(b) Again, as the index contains the only attribute of import, an index-only scan could again be performed, for a cost of 2500.

(c) This index is useless for the given query, and thus requires a sorting of the file, costing $10000 + 2500 + 2 * 3 * 2500 - 2500$ = 25000. Cost breakdown: first, we write out only the necessary attribute and do sorting pass 0, requiring reading the whole file. Then we need $\log_9(2500/10)$ = 3 passes to sort the file (sorting cost). However, instead of writing out the final merged result, in the final pass we just read the sorted runs and compute the aggregation.

(d) This is similar to the previous part, except that the initial scan requires fewer I/Os if the leaves of the B+ tree are scanned instead of the data file. $Cost = 5000 + 2500 + 2 * 3 * (2500) - 2500 = 20000$.

(e) The clustered B+ index given contains all the information required to perform an index-only scan, at a cost of $10000 *.5$.

4.
(a) Using a clustered B+ tree index on *title*, the cost of the given query is 10000 I/Os. The addition of another index would not lower the cost of any evaluation strategy that also utilizes the given index. However, the cost of the query is significantly cheaper if a clustered index on *dname, title* is available and is used by itself, and if added would reduce the cost of the best plan to 1502. (See below.)

(b) The cheapest plan here involves simply sorting the file. First we traverse the file and write out the matching records in sorted runs at a cost of at a cost of $10000 + 10000 *.25 * 10\% = 10250$ pages. For the sorting, we can do initial sorting pass using 9 buffers (because 1 is needed for scanning the file while filtering), and the merging passes using 10 buffers (1 is used for output, so the base of the log in the formula will be 9). Sorting requires $\log_9(250/9)$ = 2 passes, and we use the last pass to compute the aggregation. The total cost is 10250 + 2*2*250 – 250 = 11000.

(c) The optimal plan with the indexes given involves scanning the *dname* index and sorting the (records consisting of the) *title* field of records that satisfy the WHERE condition. This would cost $2(lookup) + 10000 * 10\%$ [retrieving qualifying records] + $10000 * 10\% *.25$ (reduction in size) [writing out *title* records] + $22 * 250 - 250$. This is a total of 2002.

(d) We can simply scan the relevant portion of the index; discard tuples that don't satisfy the WHERE condition, and write out the *title* fields of qualifying records, in total 250 pages. Then we apply the sorting and aggregation as in previous two cases. $Cost = 3(lookup) + 5000 * 10\% + 10000 * 10\% *.25 + 2 * 2 * 250 - 250 = 1503$.

(e) A clustered index on *title, dname* supports an index-only scan costing $10000 *.5 = 5000$.