CS 582 — Database Management System II— Spring 2007

Homework 4
Solution

- **11.5**

  a. The only index is on Title, and it is a clustered 2-level B+ tree.

  Solution: Because the only index is a clustered index on Title, we can use it to find the 1000*0.1=100 pages that contain employees who are programmers. The index is clustered so we can get to the first page of the data, scan through it to get E.Dept = Production. This can be done on the fly, so the result is down to 50 pages (since 5% of the employees are production programmers). This can fit in main memory. We can now project on Ename and sort in main memory to eliminate duplicates. Thus, the cost is 2 (to index) + 1 (unintegrated) or 0 (integrated) + 100 = 102 or 103 page transfers.

  b. The only index is on the attribute sequence Dept, Title, Ename; it is clustered and has two levels.

  Solution: Because we have a clustered index on Dept, Title, Ename, we can use an index only strategy. We get the top-level page of the index and from there access the index pages that contain the relevant tuples. There are 50 data pages that contain the relevant tuples, since 5% of the employees are production programmers. So, the cost of the access is at most 2 + 1 + 50 = 52/53 page I/Os. Note that since Ename is part of the key, all the production programmers will be sorted on Ename, and we will not need to do it even in main memory.

  c. The only index is on Dept, Ename, Title; it is a clustered 3-level B+ tree.

  Solution: We cannot use the index to search on Title, but we can use it to search on Dept. This will yield 1000/10=100 pages. We then follow the strategy in (a) and the cost will be 102 or 103 page transfers.

  We can also use an index-only strategy here. Search the index to find all the tuples for the production department and do projection and selection on the fly. No sorting is needed, because within each department the tuples are already sorted on Ename. The cost is 3 + the number of index pages that refer to the tuples in the production department. There are 100 pages of the corresponding data tuples, but the index is expected to be much smaller.

  d. There is an unclustered hash index on Dept and a 2-level clustered tree index on Ename.

  Solution: Searching the hash index on Dept will give us the bucket of pointers to the production employees (about 10% of the total number of tuples, 10,000). But each such pointer will result in a page fetch, since the index in unclustered. So, the cost is 1.2 I/Os to find the bucket plus 1,000 to find the relevant tuples. The above is actually slightly worse than the direct scan, which takes only 1,000 pages. The selection and projection is done on the fly and the result (50 pages) is placed in the buffer, where it is projected and then sorted to eliminate duplicates.

  Using a clustered index on Ename would require us to take a scan anyway. But since the tuples in the result are already sorted on Ename, we do not need to do any in-memory sorting, so this method is slightly preferable to the other two.

- **11.6**

  a. The fully pushed tree correspond to the following formula:

  $$Order\_by_{Budget}(\Pi_{Budget,EmpName,ProjName}(ONE \rhd\lhd_{SSN=SSN} TWO))$$

  where

  $$ONE = \Pi_{Name,SSN}(\sigma_{Name='John'}(Employee))$$

  and

  $$TWO = \Pi_{SSN,Budget,Name}(\sigma_{Budget>99}(Project))[SSN, Budget, ProjName]$$

  b. There are three reasonable plans:

  - Select Employee on Name. Since there are about 10000/1000=10 tuples in the result, it would take 1.2+10 $\sim$12 page transfers. The result contains 10 tuples, 1 page. Using index-nested loops, join the 10 Employee tuples with Project. There are two projects per employee on the average, so it would require to bring in 10 * (1.2 + 2) = 32 pages, since we will be using an unclustered hash index on SSN in Project. The result will contain 20 tuples (twice as wide as the tuples in the original relations), 1 page. We do not need to write anything out on disk, since after extracting the joined tuples we will be discarding the pages of Project that we bring in for the join. Select the result on Budget on the fly, yielding 0.01 * 20 = 1 tuple (at no cost), then project. The total cost is 12 + 32 = 44 page transfers.
  - Select Project on Budget. Because we can use a clustered 2-level index, it will take 2 I/Os to find the address of the first relevant page. Project has 20000/40 = 500 pages and we select 0.01 of them, i.e., 5 pages, 200 tuples, at the cost of 2 (index search) + 5 (reading the relevant pages) = 7 page transfers. (You can add one if the index is not integrated) Use index-nested loops to join the result with Employee. The cost: at least 200 (project tuples) * 3 (to fetch an Employee page) = 600 I/Os. The resulting number of tuples is about 200, twice the original size. This is about 10 pages. Apply selection and projection on the fly, yielding one page (at no cost). Total: over 607 I/Os.
  - Select Employee on Name, and Project on Budget. From before, we know that it takes 12 + 7 I/Os and yields 10 and 200 tuples (1 + 5 pages), respectively. Use sort-merge-join, then project. We can sort everything in memory (we have 10 pages in the buffer) and then merge, select and project in one pass (also in memory). Thus, the cost is 19 I/Os. The best plan is therefore plan #3. The second best is plan #1.