Fall, 2007 CIS 550

Database and Information Systems

Homework 6

November 26, 2007; Due December 5 at 4:30 PM

Problem 1 [25 points]:

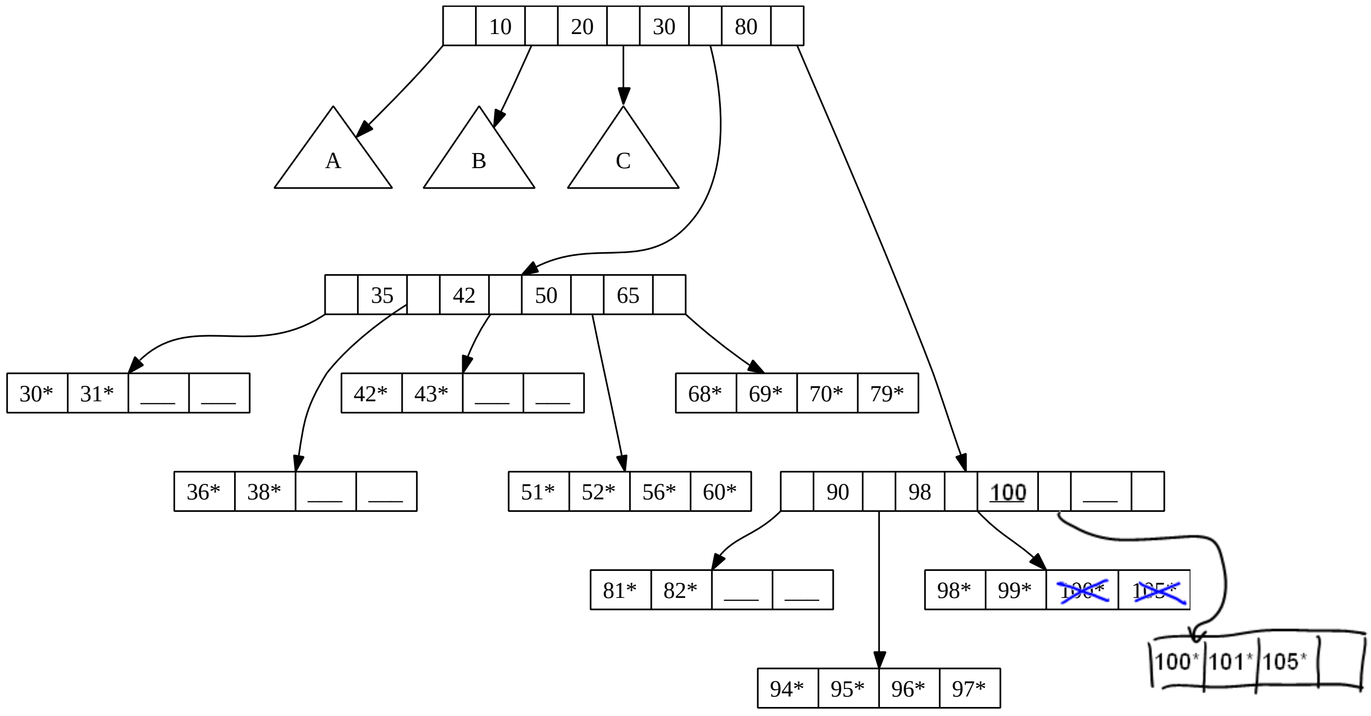

Figure 1: B+ Tree for Problem 1

Consider the B+ tree index shown in Figure 1, which stores data entries at leaf nodes in the index. Each intermediate node can hold up to five pointers and four key values. Each leaf node can hold up to four records (indicated as a key value plus a "*"), and leaf nodes are doubly linked to their predecessor and successor nodes in the index (this is not shown in the figure). The *order* of the tree, i.e., the minimum number of keys allowed in an intermediate node, is two, and the minimum number of records allowed in a leaf node is also two. Underscores ("___") indicate unallocated (free) entries.

1. Insert a record with search key 101 into the tree.

Problem 2 [50 points]: For this part of the assignment, you will need to run Oracle on eniac, using the sql command, as in previous assignments. The goal is to see the benefits of indexing in action.

The Oracle setup on eniac has a series of sample tables created for running database benchmarks — the so-called TPC-H benchmark. Additionally, each of these tables may have several alternative index structures. We will be comparing performance depending on which indices are used. To do this assignment, you will need to first enter the Oracle command **set timing on**, which will cause Oracle to report an "elapsed" time in seconds. You should run each query

The `lineitem` table has nearly 300,000 rows and takes approximately 37MB of space. We have two indices in lineitem, each with a name: `sys_c0013539` is an index over the attribute pair (l_ordernumber, l_linenumber) and `lineno` is on the singleton attribute (l_linenumber).

1. Start by running the query:

   ```
   select avg(l_linenumber)
     from zives.lineitem
    where l_linenumber > 3;
   ```

   which uses whatever indices Oracle prefers. Time the run and repeat it 5 times, averaging the total. What is that value?

   Solution:
   The average time is 0.05 sec.

2. Next, repeat the query with a special *optimizer hint* that says you want to scan the full table and avoid indices:

   ```
   select /*+ full(l) */ avg(l_linenumber)
     from zives.lineitem l
   where l_linenumber > 3;
   ```

   Average the running time for 5 runs and report the result. What can you say about the difference between the two runs? Do these results suggest that Oracle uses an index by default?

   Solution:
   The average time is 0.2 sec. When the query avoids indices, it takes about 4 times longer. It suggests Oracle use an index by default.

3. Try another hint, which makes use of the first index:

```
select /*+ index(l SYS_C001480) */ avg(l_linenumber)
  from zives.lineitem l
 where l_linenumber > 3;
```

   Run 5 times and report the result. How did this compare to the first two cases? Did that particular index offer any benefits?

   Solution:
   The average time is 0.12 sec. It is slower than the default case and faster than the no-index case. The particular index offers some benefits.

4. Which index, or indices, might have been used in the default, un-hinted Oracle query? Generalizing from the Oracle SQL hints shown above, what is the query hint that is equivalent to that default query?

   Solution:
   When use index /*+ index(l `lineno`) */, the averaged running time is about 0.05 seconds. So the default, un-hinted Oracle query might use `lineno` index.

5. For each of the two indices *and* for the non-indexed case, explain how what data and/or index structures will be examined in answering the query.

   Solution:
   The `lineno` index is sorted on l_linenumber, the system can locate the first index entry of l_linenumber >3, skip the part of the index where l_linenumber<=3. Partial indices need to be accessed.

   The `sys_c001480` index is sorted first by l_orderkey, then by l_linenumber. Two data entries with same value of l_linenumber may be located in two different places in the index tree(or index table) due to l_orderkey. Thus, the entire index has to be accessed.

   For the non-indexed case, all data entries in the lineitem relation will be scanned sequentially.

6. Run the two queries:

```
select /*+ index(l SYS_C001480) */ count(distinct l_suppkey)
  from zives.lineitem l
 where l_linenumber <= 400;
```

```
select /*+ full(l) */  count(distinct l_suppkey)
  from zives.lineitem l
 where l_linenumber <= 400;
```

Report the averaged running times over 5 runs. Explain the relative differences (or lack thereof) in performance, based on your knowledge of index and data layout characteristics, as well as your knowledge of "covering" indices.

Solution:

The first query takes about 0.72 s; the second query takes about 0.40 s.

In the first query, since `sys_c001480` doesn't contains l_supperkey, when the system finishes an index access, it has to retrieve the actual l_suppkey value by another data entry access. The physical page of the data entry may be on the different page of the index entry. Thus, to obtain a single value of l_suppkey, two disk I/Os are required and take longer time.

The second query sequentially read every data entry once.

Problem 3 [25 points]: Now we will have you specify query plans using the TPC-H benchmark data set, using Oracle. To do this assignment, you will need to repeat each query 5 times. Time it by using the Oracle command "set timing on" and reading the elapsed time. Remember to repeat the query 5 times and average the results, in order to get more consistent results.

Start with the query:

```
select count(*)
from zives.lineitem, zives.orders
where l_orderkey = o_orderkey
and o_totalprice <= 1502
```

1. What index or indices would be most useful on the lineitem and orders tables?

   Solution:

   Certainly, an index on the o_orderkey attribute would be highly useful. Possibly useful might be one on o_totalprice, but that depends on how many answers have a value less than(or equal to)1502.

2. As with before, we are going to use Oracle's *hints* to tell the optimizer how to run the query. Time the query:

   ```
   select /*+ USE_HASH(l o) */ count(*)
   from zives.lineitem l, zives.orders o
   where l_orderkey = o_orderkey
   and o_totalprice <= 1502
   ```

Solution:
Average time is 0.11 sec.

3. Replace "USE_HASH" with "USE_MERGE" and repeat.

   Solution:
   Average time is 26 sec.

4. Replace with "USE_NL" and repeat.

   Solution:
   Average time is 0.05 sec.

5. Was there any substantive difference? Which plan or plans seemed to perform best? Explain why.

   Solution:
   Yes, there is substantive difference. The nested loops join is the only one that makes use of the (default) index on o_orderkey, which is a clustered index; it's the fastest.