DB - HW 11

15.2

Consider the bank database of Figure 15.14, where the primary keys are underlined, and the following SQL query:

```
SELECT  T.branch_name
FROM    branch T, branch S
WHERE   T.assets > S.assets AND S.branch_city = "Brooklyn"
```

Write an efficient relational-algebra expression that is equivalent to this query.

Justify your answer.

Answer:

$$\Pi_{T.branch\_name}((\Pi_{T.branch\_name,\, assets}(\rho_T(branch))) \bowtie_{T.assets > S.assets} (\Pi_{assets}(\sigma_{branch\_city='Brooklyn'}(\rho_S(branch))))$$

We do the selection $\sigma_{branch\_city='Brooklyn'}$ before doing the Theta Join, which eliminated irrelevant tuples and thus significantly reduced the number of data involved in this process.

15.3

Let relations r_1 (A,B,C) and r_2(C,D,E) have the following properties: r_1 has 20,000 tuples, r_2 has 45,000 tuples, 25 tuples of r_1 fit on one block, and 30 tuples of r_2 fit on one block. Estimate the number of block transfer and seeks required using each of the following join strategies for $r_1 \bowtie r_2$:

a. Nested-loop join.

b. Block nested-loop join.

c. Merge join.

d. Hash join.

Answer:

r_1 needs $20000 \div 25 = 800$ blocks

r_2 needs $45000 \div 30 = 1500$ blocks

Assuming the memory has N pages：

a. Nested-loop Join

- Use r_1 as outer relation： $20000 * 1500 + 800 = 30000800$
- Use r_2 as outer relation： $45000 * 800 + 1500 = 36001500$

b. Block nested-loop join

- Use r_1 as outer relation： $\lceil \frac{800}{N-1} \rceil \ * \ 1500 \ + \ 800$
- Use r_2 as outer relation： $\lceil \frac{1500}{N-1} \rceil \ * \ 800 \ + \ 1500$

c. Merge join

If both of r_1 and r_2 are not sorted on the join key, the cost of sorting would be:

$$1500 \ * \ (2 \ * \ \lceil log_{N-1}(\frac{1500}{N})\rceil \ + \ 2) \ + \ 800 \ * \ (2 \ * \ \lceil log_{N-1}(\frac{800}{N}) \ + \ 2)$$

Assuming all the tuples possessing the same value for join attributes fit in memory, the total cost would be:

$$1500 \ * \ (2 \ * \ \lceil log_{N-1}(\frac{1500}{N})\rceil \ + \ 2) \ + \ 800 \ * \ (2 \ * \ \lceil log_{N-1}(\frac{800}{N}) \ + \ 2) \ + \ 1500 \ + \ 800$$

d. Hash join

Assuming there is no flow, using r_1 as build relation ad r_2 as probe relation:

If $N \ > \ \frac{800}{N}$:

As there is no need to do recursive partitioning, the cost would be $3 \ * \ 1500 \ + \ 800 = 6900$

Else:

The cost would be $2 \ * \ (1500 \ + \ 800) * \lceil log_{N-1}(800) \ - \ 1 \rceil \ + \ 1500 \ + \ 800$

15.6

Consider the bank database of Figure 15.14, where the primary keys are underlined. Suppose that a B^+-tree index on *branch_city* is available on relation *branch*, and that no other index is available. List different ways to handle the following selections that involve negation:

a. $\sigma_{\neg(branch\_city < ''Brooklyn'')}(branch)$

b. $\sigma_{\neg(branch\_city = ''Brooklyn'')}(branch)$

c. $\sigma_{\neg(branch\_city < ''Brooklyn'' \vee assets < 5000)}(branch)$

Answer:

a. $\because \neg(branch\_city <' Brooklyn') \Longleftrightarrow branch\_city \geqslant \ 'Brooklyn'$

Thus, we just need to locate the first tuple meeting the requirement: record.branch_city = 'Brooklyn' by using the index, and then retrieve all the tuple from this position to the end by following the pointer chains.

b. $\because \neg(branch\_city =' Brooklyn') \Longleftrightarrow branch\_city \neq \ 'Brooklyn'$

In this case, scan the whole file sequentially performs better than using the index. Thus, we can simply scan the whole file in sequential order and retrieve all the tuple whose value of field *branch_city* is anything but 'Brooklyn'.

c.
$\because \neg(branch\_city <' Brooklyn' \ \vee \ assets < 5000) \Longleftrightarrow branch\_city \geqslant \ 'Brooklyn' \wedge assets \ \geqslant \ 5000$

Thus, like what we did in a), we can find the first tuple whose value of branch_city equals to 'Brooklyn' by using the index of branch_city, and follow the pointer chain to retrieve every tuple whose value of attribute *branch_city* is bigger than or equal to 'Brooklyn'. Additionally, we should check whether the value of attribute *assets* we visit is no less than 5000.

Or we can find the first tuple whose value of attribute *assets* is no less than 5000 and retrieve all the tuples till the end of file. Besides, checking whether the value of attribute branch_city is bigger than or equal to 'Brooklyn'.

15.20

Estimate the number of block transfers and seeks required by your solution to Exercise 15.19 for $r_1 \bowtie r_2$ are as defined in Exercise 15.3

Answer:

In Exercise 15.3 we have already calculated that r_1 and r_2 takes 800 and 1500 blocks respectively.

Assuming there are n pointers in each leaf block of an index, then:

index of r_1 needs $\lceil \frac{20000}{n} \rceil$ leaf blocks, and index of r_2 needs $\lceil \frac{45000}{n} \rceil$ leaf blocks.

Thus, without output, the merge sort would take $\lceil \frac{20000}{n} \rceil + \lceil \frac{45000}{n} \rceil$ times of access.

The number of output tuples = $\lceil \frac{20000*45000}{max(V(C,r_1),V(C,r_2))} \rceil$, each of them would take 2 pointers.

The number of blocks for output of join $= \lceil \frac{\lceil \frac{20000*45000}{max(V(C,r_1),V(C,r_2))} \rceil}{n/2} \rceil = \lceil \frac{2}{n} * \lceil \frac{20000*45000}{max(V(C,r_1),V(C,r_2))} \rceil\rceil$

The total disc block accesses of the join $= \lceil \frac{2}{n} * \lceil \frac{20000*45000}{max(V(C,r_1),V(C,r_2))} \rceil\rceil + \lceil \frac{20000}{n} \rceil + \lceil \frac{45000}{n} \rceil$

Assuming the memory has M pages, when replacing the pointers by actual tuples, we needs:

$\lceil \frac{2}{n} * \lceil \frac{20000*45000}{max(V(C,r_1),V(C,r_2))} \rceil\rceil * (2 + 2 * log_{M-1}(\frac{\lceil \frac{2}{n} * \lceil \frac{20000*45000}{max(V(C,r_1),V(C,r_2))} \rceil\rceil}{M}))$ disc block accesses for the first sorting.

Besides, there are $min(800, \lceil \frac{20000*45000}{max(V(C,r_1),V(C,r_2))} \rceil)$ blocks in r_1 whose pointer should be replaced.

Assuming that in 1 disk block, there are K_1 pairs of (tuple_of_r_1, ptr_to_r_2), thus the intermediate result would take $\lceil \frac{\lceil \frac{20000*45000}{max(V(C,r_1),V(C,r_2))} \rceil}{K_1} \rceil$ blocks.

The first pass of replacement will cost:

$$\lceil \frac{2}{n} * \lceil \frac{20000*45000}{max(V(C,r_1),V(C,r_2))} \rceil\rceil * (2 + 2 * log_{M-1}(\frac{\lceil \frac{2}{n} * \lceil \frac{20000*45000}{max(V(C,r_1),V(C,r_2))} \rceil\rceil}{M})) + \lceil \frac{2}{n} * \lceil \frac{20000*45000}{max(V(C,r_1),V(C,r_2))} \rceil$$
$$+ min(800, \lceil \frac{20000*45000}{max(V(C,r_1),V(C,r_2))} \rceil) + \lceil \frac{\lceil \frac{20000*45000}{max(V(C,r_1),V(C,r_2))} \rceil}{K_1} \rceil \qquad times\ of\ \ block\ accesses$$

Likewise, assuming that there are K_2 pairs of (tuple_of_r_2, ptr_to_r_1) in 1 block,.

The second pass of replacement will cost:

$$\lceil \frac{\lceil \frac{20000*45000}{max(V(C,r_1),V(C,r_2))} \rceil}{K_1} \rceil * (2 + 2 * log_{M-1}(\frac{\lceil \frac{\lceil \frac{20000*45000}{max(V(C,r_1),V(C,r_2))} \rceil}{K_1} \rceil}{M}))$$
$$+ \lceil \frac{\lceil \frac{2}{n} * \lceil \frac{20000*45000}{max(V(C,r_1),V(C,r_2))} \rceil\rceil}{K_1} \rceil + min(1500, \lceil \frac{20000*45000}{max(V(C,r_1),V(C,r_2))} \rceil)$$

By adding up all the three parts, we can have the total number of disc access for the join:

$$
\lceil\frac{20000}{n}\rceil + \lceil\frac{45000}{n}\rceil + 2 * \lceil\frac{\lceil\frac{20000*45000}{max(V(C,r_1),V(C,r_2))}\rceil}{K_1}\rceil * (2 + log_{M-1}(\frac{\lceil\frac{\lceil\frac{20000*45000}{max(V(C,r_1),V(C,r_2))}\rceil}{K_1}\rceil}{M}))
$$

$$
+ 2 * \lceil\frac{2}{n} * \lceil\frac{20000 * 45000}{max(V(C,r_1),V(C,r_2))}\rceil\rceil * (2 + log_{M-1}(\frac{\lceil\frac{2}{n} * \lceil\frac{20000*45000}{max(V(C,r_1),V(C,r_2))}\rceil\rceil}{M}))
$$

$$
+ min(1500, \lceil\frac{20000 * 45000}{max(V(C,r_1),V(C,r_2))}\rceil) + min(800, \lceil\frac{20000 * 45000}{max(V(C,r_1),V(C,r_2))}\rceil)
$$

And the number of output pages $= \lceil\frac{\lceil\frac{20000*45000}{max(V(C,r_1),V(C,r_2))}\rceil}{K_2}\rceil$