Sheet03

Answers to Selected Exercises

15.13 Consider SQL queries Q1, Q8, Q1B, Q4, Q27 from Chapter 8.

(a) Draw at least two query trees that can represented each of these queries. Under what circumstances would you use each of your query trees?

(b) Draw the initial query tree for each of these queries; then show how the query tree is optimized by the algorithm outlined in section 15.7.

(c) For each query, compare your on query trees of part (a) and the initial and final query trees of part (b).

Answer:

Below are possible answers for Q8 and Q27.

Q8: SELECT E.FNAME, E.LNAME, S.FNAME, S.LNAME
FROM EMPLOYEE E, EMPLOYEE S
WHERE E.SUPERSSN = S.SSN

Q8's tree1:

PROJECT E.FNAME, E.LNAME, S.FNAME, S.LNAME
JOIN E.SUPERSSN=S.SSN
EMPLOYEE E EMPLOYEE S

Q8'S tree2:

PROJECT E.FNAME, E.LNAME, S.FNAME, S.LNAME
SELECT E.SUPERSSN=S.SSN
CARTESIAN PRODUCT
EMPLOYEE E EMPLOYEE S

The initial query tree for Q8 is the same as tree2 above; the only change made by the optimization algorithm is to replace the selection and Cartesian product by the join in tree1. Thus, tree 1 is the result after optimization.

Q27: SELECT FNAME, LNAME, 1.1*SALARY
FROM EMPLOYEE, WORKS_ON, PROJECT
WHERE SSN = ESSN AND PNO = PNUMBER AND PNAME = 'ProductX'

Q27's tree1:

PROJECT FNAME, LNAME, SALARY
SELECT PNAME="ProductX"
JOIN SSN=ESSN
JOIN PNO=PNUMBER EMPLOYEE
PROJECT WORKS_ON

Q27's tree2:

PROJECT FNAME, LNAME, SALARY
SELECT PNO=PNUMBER AND SSN=ESSN AND PNAME="ProductX"
CARTESIAN PRODUCT
CARTESIAN PRODUCT EMPLOYEE
WORKS_ON PROJECT

The initial query tree of Q27 is tree2 above, but the result of the heuristic optimization process will NOT be the same as tree1 in this case. It can be optimized more thoroughly, as follows:

PROJECT FNAME, LNAME, SALARY
JOIN SSN=ESSN
JOIN PNO=PNUMBER EMPLOYEE
SELECT PNAME="ProductX"
PROJECT WORKS_ON

15.14 A file of 4096 blocks is to be sorted with an available buffer space of 64 blocks. How many passes will be needed in the merge phase of the external sort-merge algorithm?

Answer:

Number of passes: $1 + \lceil \log_{B-1} \lceil N/B \rceil \rceil$

We first need to compute the number of runs,

Where:

$N = 4096$ is the number of blocks in file

$B = 64$ buffer blocks

$$\text{Number of passes} = 1 + \left\lceil \log_{63} \left\lceil \frac{4096}{64} \right\rceil \right\rceil = 1 + 2 = 3$$

15.15 Develop (approximate) cost functions for the PROJECT, UNION, INTERSECTION, SET DIFFERENCE, and CARTESIAN PRODUCT algorithms

Answer:

Assume relations R and S are stored in bR and bS disk blocks, respectively. Also, assume that the file resulting from the operation is stored in bRESULT disk blocks (if the size cannot be otherwise determined).

PROJECT operation:

if <attribute list> includes a key of R, then the cost is **2*bR** since the read–in and write–out files have the same size, which is the size of R itself;

if <attribute list> does not include a key of R, then we must sort the intermediate result file before eliminating duplicates, which costs another scan; thus the latter cost is **3*bR + bR *log$_2$ bR** (assuming PROJECT is implemented by sort and eliminate duplicates).

SET OPERATIONS (**UNION, DIFFERENCE, INTERSECTION**): According to the algorithms where the usual sort and scan/merge of each table is done,

the cost of any of these is:
(bR *log$_2$ bR) + (bS *log$_2$ bS) + bR + bS + bRESULT

CARTESIAN PRODUCT:

The join selectivity of Cartesian product is = 1, and the typical way of doing it is the nested loop method, since there are no conditions to match. We first assume two memory buffers; We get:

J1: C R X S = bR + (bR *bS) + (|R|*|S|)/bfr(R*S)

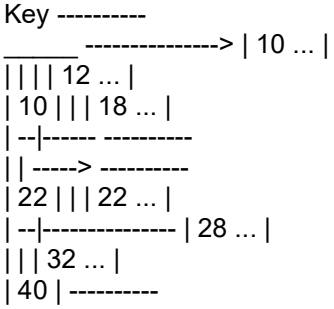
Next, suppose we have nB memory buffers, Assume file R is smaller and is used in the outer loop. We get:

J1': C R X S = bR + (ceiling(bR /(nB - 1)) * bS)+ (|R|*|S|)/bfr (R*S) , which is better than
J1, per its middle term.

15.17 Can a nondense (sparse) index be used in the implementation of an aggregate operator? Why or why not?

Answer:

A nondense (sparse) index contains entries for only some of the search values. A primary index is an example of a nondense index which includes an entry for each disk block of the data file rather than for every record.



| --|------
|____| | ----------
----------> | 40 ... |
| 52 ... |
| 60 ... |

If the keys in the index correspond to the smallest key value in the data block then the sparse index could be used to compute the MIN function. However, the MAX function could not be determined from the index. In addition, since all values do not appear in the index, AVG, SUM and COUNT could not be determined from just the index.

15.21 Extend the sort-merge join algorithm to implement the left outer join.

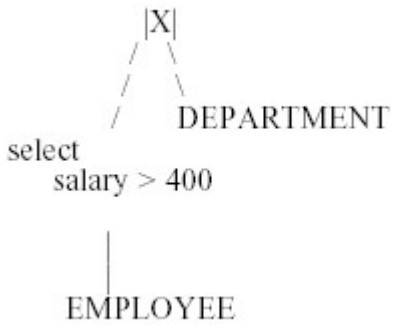
Answer:

The left outer join of R and S would produce the rows from R and S that join as well as the rows from R that do not join any row from S. The sort-merge join algorithm would only need minor modifications during the merge phase. During the first step, relation R would be sorted on the join column(s).
During the second step, relation S would be sorted on the join columns(s).
During the third step, the sorted R relation and S relation would be merged. That is, rows would be combined if the R row and S row have the same value for the join column(s).
In addition, if no matching S row was found for an R row, then that R row would also be placed in the left outer join result, except that the corresponding attributes from the S relation would be set to NULL values. (How a null value is to be represented is a matter of choice out of the options provided in a DBMS.)

15.22 Compare the cost of two different query plans for the following query:
salary > 40000 select (EMPLOYEE |X| DNO=DNUMBER DEPARTMENT)
Use the database statistics in Figure 15.8

Answer:

One plan might be for the following query tree



We can use the salary index on EMPLOYEE for the select operation:

The table in Figure 15.8(a) indicates that there are 500 unique salary values, with a low value of 1 and a high value of 500. (It might be in reality that salary is in units of 1000 dollars, so 1 represents $1000 and 500 represents $500,000.)
The selectivity for (Salary > 400) can be estimated as
(500 - 400)/500 = 1/5

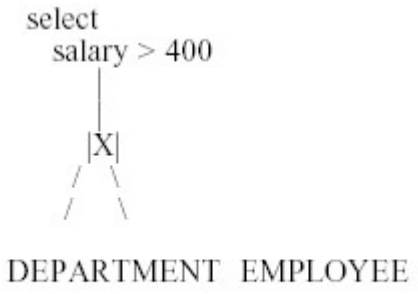
This assumes that salaries are spread evenly across employees.

So the cost (in block accesses) of accessing the index would be

Blevel + (1/5) * (LEAF BLOCKS) = 1 + (1/5)* 50 = 11

Since the index is nonunique, the employees can be stored on any of the data blocks.
So the the number of data blocks to be accessed would be
(1/5) * (NUM_ROWS) = (1/5) * 10,000 = 2000

Since 10,000 rows are stored in 2000 blocks, we have that
2000 rows can be stored in 400 blocks. So the TEMPORARY table (i.e., the result of the selection operator) would contain 400 blocks.
The cost of writing the TEMPORARY table to disk would be
400 blocks.
Now, we can do a nested loop join of the temporary table and the DEPARTMENT table. The cost of this would be
b + (b * b)
DEPARTMENT DEPARTMENT TEMPORARY
We can ignore the cost of writing the result for this comparision, since the cost would be the same for both plans, we will consider.
We have 5 + (5 * 400) = 2005 block accesses
18.22 (continued)
Therefore, the total cost would be
11 + 2000 + 400 + 2005 = 4416 block accesses
NOTE: If we have 5 main memory buffer pages available during the join, then we could store all 5 blocks of the DEPARTMENT table there. This would reduce the cost of the join to 5 + 400 = 405 and the total cost would be reduced to 11 + 2000 + 400 + 405 = 2816. A second plan might be for the following query tree



Again, we could use a nested loop for the join but instead of creating a temporary table for the result we can use a pipelining approach and pass the joining rows to the select
operator as they are computed. Using a nested loop join algorithm would yield the following
5 + (5 * 2000) = 100,050 blocks We would pipeline the result to the selection operator
and it would choose only those rows whose salary value was greater than 400.
NOTE: If we have 5 main memory buffer pages available during the join, then we could store the entire DEPARTMENT table there. This would reduce the cost of the join and the pipelined select to 50 + 2000 = 2050.