Implementation of Databases
Assignment #3

Due on November 29, 2016

Prof. Dr. rer. pol. Matthias Jarke

Dr. rer. nat. Christoph Quix

Submitted by:

Sanchit Alekh, Idil Esen Zulfikar

MSc. Software Systems Engineering

Problem 1

Part 1

Let R be a relation of attributes A, B and C. Given is a query in relational algebra (RA).

(a) Translate the RA query into an equivalent query in Domain Relational Calculus (DRC).

Solution

$< A_1, B_1, C_1 >| \exists A_2, B_2, C_2, A_3, B_3, C_3$
$R_3(A_3, B_3, C_3), R_2(A_2, B_2, C_2), R_1(A_1, B_1, C_1)$
$\wedge (A_2 = A_3) \wedge (B_1 = B_2) \wedge (B_2 = 1)$

(b) Construct an equivalent tableau of the query.

Solution

a_1	b_1	c_1	
a_1	b_1	c_1	R
a_2	1	c_2	R
a_2	b_3	c_3	R
$b_1 = 1$			

Table 1: Tableau Representation of the Given Relational Algebra Query

Part 2

Given the following two tableau queries T1 and T2, decide whether they are equivalent, subsumed by each other, or not related at all.

Solution

Checking for $T_1 \subseteq T_2$, looking for mappings h from T_2 to T_1
$h(a_5) = a_1$
$h(b) = b$
$h(c_4) = c_1$

$h(a) = a$
$h(b_5) = b_1$
$h(c_4) = c_1$
Since a mapping h exists for each element $e \in T_2$, $T_1 \subseteq T_2$

Problem 1 continued on next page...

Checking for $T_2 \subseteq T_1$, looking for mappings *h'* from T_1 to T_2
$h'(a) = a$
$h'(b_1) = b_5$
$h'(c_1) = c_4$

$h'(a) = a$
$h'(b_3) = b_5$
$h'(c_3) = c_4$

$h'(a_1) = a_5$
$h'(b) = b$
$h'(c_1) = c_4$

$h'(a_1) = a_5$
$h'(b_2) = b$
$h'(c_2) = c_4$

Since a mapping h' exists for each element $e' \in T_1$, $T_2 \subseteq T_1$
We have proved that $T_1 \subseteq T_2$ and $T_2 \subseteq T_1$
Therefore T_1 and T_2 are equivalent to each other.

Problem 2

Given is the following relational database schema (keys are underlined):

1. **TRC**:
   Specify the following query in the tuple relational calculus (TRC) and draw the corresponding quant graph: Find the names of the chefs, who created the brand, and the shops of his brand are located in the same city where the chef was born.

   ## Solution

   $c.ChefName \mid c \in Chef \wedge \exists cr \in Create$
   $cr.CID = c.CID \wedge$
   $\exists s \in Shop \wedge s.BID = cr.BID \wedge s.City = c.CityOfBirth$
   The Quant Graph is shown in *Figure 1*.

2. Does the graph contain a cycle? What does the result mean for optimization?

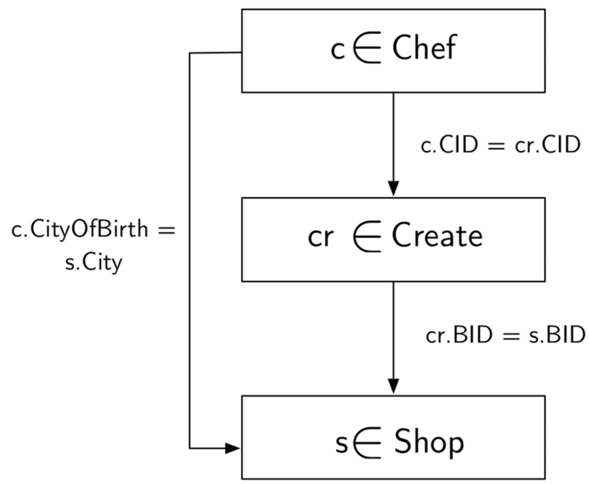

Figure 1: Quant Graph representation of the Query

Solution

Yes, the quant graph in this question contains a cycle.
This means that the query implementation will be extremely sub-optimal no matter what join order is chosen. This is because the semi-joins give intermediate tuples which are much larger than the desired result. This makes it really difficult to employ any query optimization strategy.

Problem 3

Consider the following relational schema and SQL query. The schema captures information about employees, departments, and projects.

1. Compute the number of pages for each relation.

Solution

For Emp Relation:

- Total number of tuples = 20000
- Size of each tuple = 20 bytes
- Size of each page = 4000 bytes
- Hence, no. of pages = $\frac{total\ size\ of\ all\ tuples}{size\ of\ one\ page} = \frac{20000 \times 20}{4000} = 100$ pages

For DeptProj Relation:

- Total number of tuples = 5000
- Size of each tuple = 40 bytes
- Size of each page = 4000 bytes
- Hence, no. of pages = $\frac{total\ size\ of\ all\ tuples}{size\ of\ one\ page} = \frac{5000 \times 40}{4000} = 50$ pages

For Proj Relation:

- Total number of tuples = 1000
- Size of each tuple = 2000 bytes
- Size of each page = 4000 bytes
- Hence, no. of pages = $\frac{total\ size\ of\ all\ tuples}{size\ of\ one\ page} = \frac{2000 \times 1000}{4000} = 500$ pages

2. Consider the following query: Find all employees with age more than 30." Assume that there is an unclustered index on age. Let the number of qualifying tuples be N. For what values of N is a sequential scan cheaper than using the index?

Solution

Cost of page-by-page access for the *Emp* relation = 100 I/Os, since there are 100 pages
Cost of I/Os using the unclustered index = Cost of index access + Page I/O for each qualifying tuple = 3 + N
Therefore, for sequential scan to be cheaper than using the index

$$3 + N > 100$$
$$\therefore\ N > 97$$

Therefore, $N > 97$ for sequential scan to be cheaper.

3. Consider the following query:

SELECT * FROM Emp E, DeptProj D WHERE E.did=D.did

a Compute the costs for the query using a block-nested loop join.

Solution

For a block-nested loop join, the costs are given by:

$$M + \lceil \frac{M}{B-2} \rceil \times N$$
$$= 50 + \lceil \frac{50}{12-2} \rceil \times 100 = 50 + 500 = 550 I/Os$$

b Suppose that there is a clustered hash index on did on Emp. Compute the costs for the query using an index-nested loop join.

Solution

If a clustered hash index is available on *Emp*, the best approach is to make it the inner relation and *DeptProj* the outer relation to use the Hash Index.
Cost of scanning the outer relation (DeptProj) = Number of pages = 50 I/O
Now, for each tuple in DeptProj, we have the following costs:

- Cost of Index Access = 3 I/Os
- Cost of retrieving the matching tuple = 1 I/O

Therefore, total cost:
$M + t_M \times$ (Cost of Index Access + Cost of Retrieving Matching Tuples)
$= 50 + 5000 \times (3 + 1) = 50 + 5000 \times 4 = 20050$ I/Os

c Assume that both relations are sorted on the join column. Which join method should be applied and what are the costs?

Solution

Since both relations are sorted, the sort-merge join is appropriate. The cost of Sort-merge join is calculated as the cost of sorting the relations on the join column + merging them, i.e. $O(MlogM) + O(NlogN) + M + N$ I/Os. In this case, both relations are already sorted on the join column, so the only thing should be calculated is the merge operation, which equals $M + N$.

$\therefore$ The cost of joining $= M + N = 100 + 50 = 150 I/Os$

d Suppose that there is a clustered B+ tree index on did on Emp and DeptProj is sorted on did. Which join method should be applied and what are the costs?

Solution

$\because$ There is a clustered B+ tree index on *Emp* on the key *did*, the values in the *Emp* relation must be sorted on *did*.
Furthermore, *DeptProj* is also sorted on *did*.
$\therefore$ The most efficient approach is to use the sort-merge join, since both the relations are already sorted on the merge column.
The only costs to consider, would be the costs for joining, i.e. M + N

$$\therefore \text{Total Cost for the join} = 100 + 50 = 150 \text{ I/Os.}$$