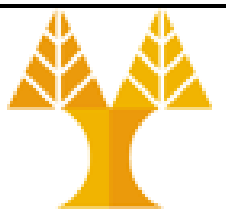

ΕΡΓΑΣΤΗΡΙΟ #6

QUERY OPTIMIZER

1. **(Exercise 15.3)** For each of the following SQL queries, for each relation involved, list the attributes that must be examined to compute the answer. All queries refer to the following relations:

   Emp(*eid:* integer, *did:* integer, *sal:* integer, *hobby:* char(20))
   Dept(*did:* integer, *dname:* char(20), *floor:* integer, *budget:* real)

   a. SELECT COUNT(*) FROM Emp E, Dept D WHERE E.did = D.did
   b. SELECT MAX(E.sal) FROM Emp E, Dept D WHERE E.did = D.did
   c. SELECT MAX(E.sal) FROM Emp E, Dept D WHERE E.did = D.did AND D.floor = 5
   d. SELECT E.did, COUNT(*) FROM Emp E, Dept D WHERE E.did = D.did GROUP BY D.did
   e. SELECT D.floor, AVG(D.budget) FROM Dept D GROUP BY D.floor HAVING COUNT(*) > 2
   f. SELECT D.floor, AVG(D.budget) FROM Dept D GROUP BY D.floor ORDER BY D.floor

   Answer:
   a. E.did, D.did
   b. E.sal, E.did, D.did
   c. E.sal, E.did, D.did, D.floor
   d. E.did, D.did
   e. D.floor, D.budget
   f. D.floor, D.budget

2. **(Exercise 15.7)** Consider the following relational schema and SQL query. The schema captures information about employees, departments, and company finances (organized on a per department basis).

   Emp(eid: integer, did: integer, sal: integer, hobby: char(20))
   Dept(did: integer, dname: char(20), floor: integer, phone: char(10))
   Finance(did: integer, budget: real, sales: real, expenses: real)

   Consider the following query:

   ```
   SELECT D.dname, F.budget
   FROM Emp E, Dept D, Finance F
   WHERE E.did=D.did AND D.did=F.did AND D.floor=1
   AND E.sal ≥ 59000 AND E.hobby = 'yodeling'
   ```

   a. Identify a relational algebra tree (or a relational algebra expression if you prefer) that reflects the order of operations a decent query optimizer would choose.
   b. List the join orders (i.e., orders in which pairs of relations can be joined to compute the query result) that a relational query optimizer will consider. (Assume that the optimizer follows the heuristic of never considering plans that require the computation of cross-products.) Briefly explain how you arrived at your list.
   c. Suppose that the following additional information is available: Unclustered B+ tree indexes exist on Emp.did, Emp.sal, Dept.floor, Dept.did, and Finance.did. The system's statistics indicate that employee salaries range from 10,000 to 60,000, employees enjoy 200 different hobbies, and the company owns two floors in the building. There are a total of 50,000 employees and 5,000 departments (each with corresponding financial information) in the database. The DBMS used by the company has just one join method available, index nested loops.
      i. For each of the query's base relations (Emp, Dept, and Finance) estimate the number of tuples that would be initially selected from that relation if all of the non-join predicates on that relation were applied to it before any join processing begins.
      ii. Given your answer to the preceding question, which of the join orders considered by the optimizer has the lowest estimated cost?

   Answer:

   a. $\pi_{D.dname,F.budget}(((\pi_{E.did}\left(\sigma_{E.sal\geq 59000,E.hobby="yodelling"}(E)\right) \bowtie \pi_{D.did,D.dname}(\sigma_{D.floor=1}(D))) \bowtie \pi_{F.budget,F.did}(F))$

   b. There are 2 join orders considered, assuming that the optimizer only consider left-deep joins and ignores cross-products: (D,E,F) and (D,F,E)

   c.

      i. The answer to each relation is given below.
      *Emp: 50000 records, E.sal ≥ 59,000, E.hobby = "yodelling", result = 50000 * 1/50 * 1/200 = 5*
      *Dept: 5000 records, D.floor = 1, result = 5000 * 1/2 = 2500*
      Finance: 5000 records, there are no non-join predicates, result = 5000

      ii. *Consider the following join methods on the following left-deep tree: ((E |x| D) |x| F). The tuples from E will be pipelined, no temporary relations are created. First, retrieve the tuples from E with salary ≥ 59,000 using the B-tree index on salary; we estimate 1000 such tuples will be found, with a cost of 1 tree traversal + the cost of retrieving the 1000 tuples (since the index is unclustered) = 3+1000 = 1003. Note, we ignore the cost of scanning the leaves.*

Of these 1000 retrieved tuples, on the fly select only those that have hobby = ”yodelling”, we estimate there will be 5 such tuples. Pipeline these 5 tuples one at a time to D, and using the B-tree index on D.did and the fact the D.did is a key, we can find the matching tuples for the join by searching the Btree and retrieving at most 1 matching tuple, for a total cost of 5(3 + 1) = 20. The resulting cardinality of this join is at most 5. Pipeline the estimated 3 tuples of these 5 that have D.floor=1 1 up to F, and use the Btree index on F.did and the fact that F.did is a key to retrieve at most 1 F tuple for each of the 3 pipelined tuples. This costs at most 3(3+1) = 12.
Ignoring the cost of writing out the final result, we get a total cost of 1003+20+12 = 1035.