Practice Problem Set #8

CSC 261/461 (Database Systems), Spring 2018,
University of Rochester
04/04/2018

Sections to study

Note: There is no better alternative than studying the textbook! Reading the following sections would reinforce your understanding of the material.

Chapter 18

- 18.4.1 (J3 and J4)
- 18.4.4
- Section 19.1

Chapter 19

- Very Important: 19.1.2. We have gone through the material but you must study this section for better understanding.
  If must understand each and every step in Figure 19.2 is performed. Study the last paragraph: Summary of Heuristics for Algebraic Optimization.
- 19.3.2
- 19.3.3
- 19.4 (S1 to S6)
- 19.5 (J1-J4) (Important. We cover the exact same topic using easier representation.)

Chapter 20

- Studying only the slides is sufficient.

Problem 1

Assume an Employee table Employee(EmployeeID, Salary) is provided. The total number of records in Employee table is 60,000. The size of each record is 20 bytes. Assume, the size of each block is 200 bytes.

Assume an **equi-height** (practice with equi-weight too) histogram is provided, where the buckets are:
10k-15k, 15k-20k, 20k-30k, 30k-50k, 50k-80k.
Each bucket (x-y) stores the count of employees where **salary >= x and salary < y**. Also assume, x and y are multiples of 1000. Assume, each EmployeeID is unique.

Answer the following questions.
(Note: Each question is independent. Any assumptions made are local to that question.)

Solution: Number of record, r = 60000
Number record per block, bfr = 200 /20 = 10
Number of total block, b = 60000 /10 = 6000
Each bucket contains 60,000 / 5 = 12000 records

1. What is the average cost to find the employee with EmployeeID = "123456"?

   Solution: Cost = b/2 = 6000/2 = 3000

2. What is the cost to find the employees whose salary is 40k using binary search? Assume, the entries are sorted based on salary.

   Solution:
   Cost = $\log_2 b + \lceil s/bfr \rceil - 1$ = $\log_2 6000$ + 600 - 1 = 12.55 + 599 $\approx$ 612
   Selection cardinality, s = 12000 / 20 = 6000 (Bucket Size / Bucket Entries for Bucket 30k-50k)

3. Assume, we have primary index on EmployeeID. Number of levels for this index is 3. What is the cost to find an employee with EmployeeID = "123456"?

   Solution: Cost = x + 1 = 4

4. Assume, we have an ordering index on Salary. What is the cost to find all the employees whose salary is greater than 40k? Assume, number of level for this index is 3.

   Solution: Cost = x + b * (1.5 / 5) = 3 + 6000 * 1.5 / 5 = 1803
   (1.5 buckets out of 5 satisfy the condition. Why 1.5? Because: We need to consider one full bucket: (50k-80k) and the other 0.5 bucket comes from: (30k-50k) bucket. On an average, in this bucket, half of the records will have value greater than 40k)

5. Assume, we have a secondary B+ index on Salary with 3 level. What is the cost to find all the employees whose salary is 40k?

   Solution: Cost = x + 1 + s = 3 + 1 + 6000 = 6004
   Selection cardinality, s = 12000 / 20 = 6000 (Bucket Size / Bucket Entries for Bucket 30k-50k)

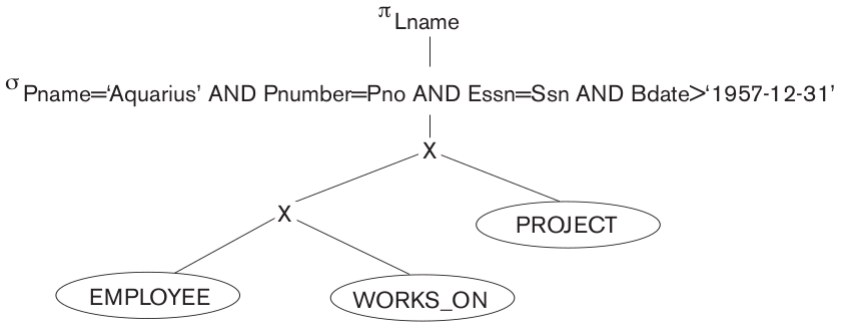
Problem 2 (Fig: 19.2)

Consider the following query.

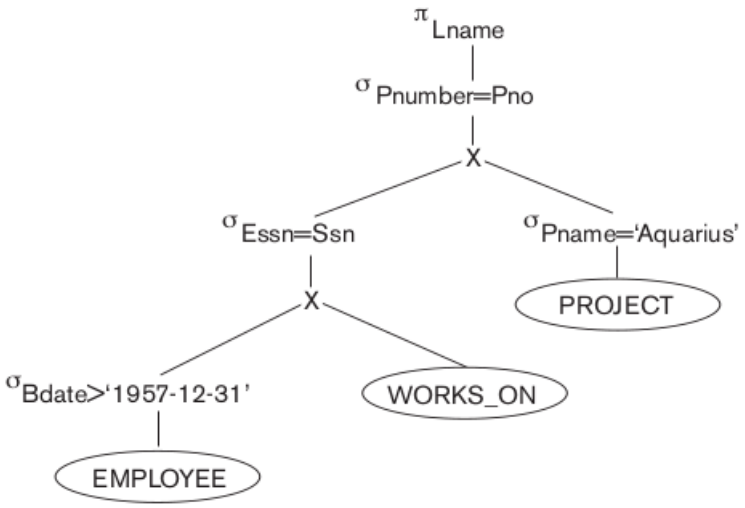
1. Draw the initial (canonical) query tree for SQL query

```
SELECT  Lname
FROM    EMPLOYEE, WORKS_ON, PROJECT
WHERE   Pname='Aquarius' AND Pnumber=Pno AND Essn=Ssn
        AND Bdate > '1957-12-31';
```

Solution: initial (canonical): We will just use our basic Cartesian product. No optimization at all. Also, we will maintain the order of the relations. In our case: Employee, Works_on, Project (in order)



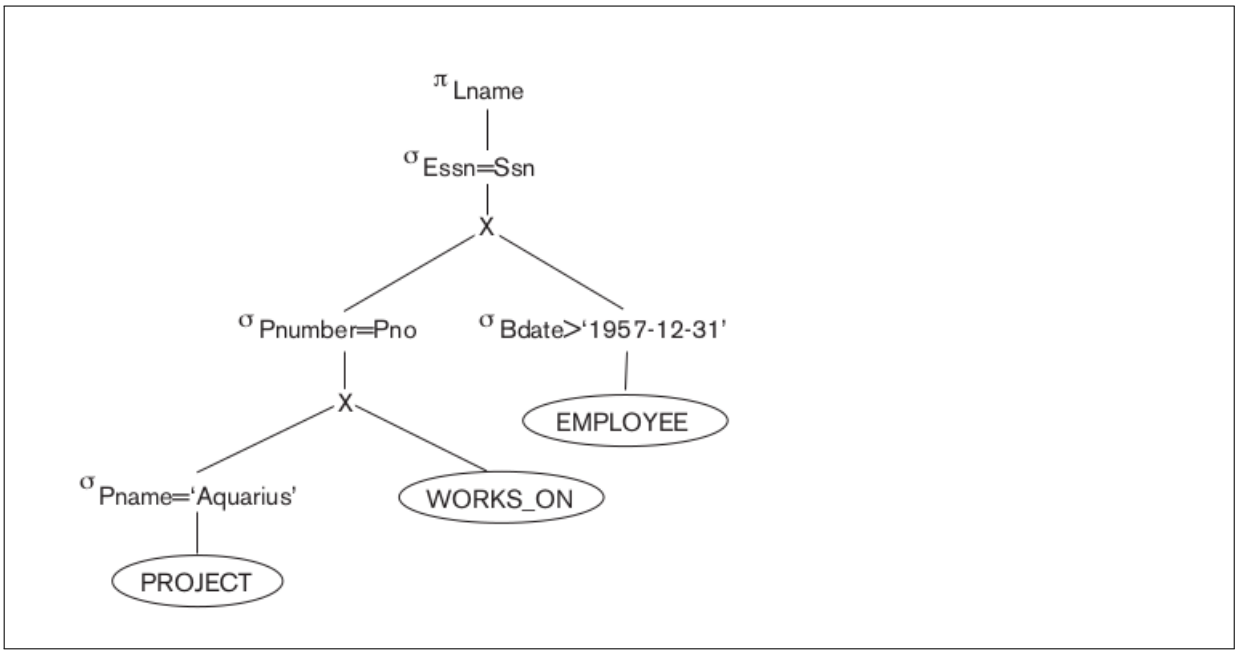
2. Move SELECT operations down the query tree.



Solution:

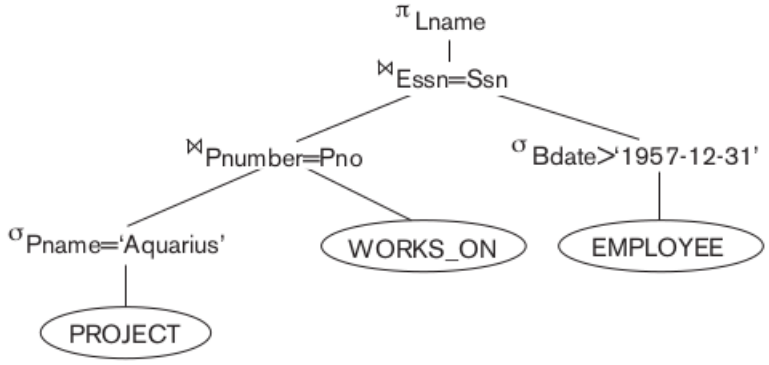
3. Apply the more restrictive SELECT operation first.

Solution: This one is important. Assuming, Pname is the unique key in Project table, we know that Pname = "Aquarius" will return only one tuple. So, this selection is the most restrictive one. So, start with this. See, how we have changed the order of the relations



4. Replace CARTESIAN PRODUCT and SELECT with JOIN operations

Solution: Now, replace the cartesian product and select with Join.
When we are asking for canonical query tree, the tree should not have join operation if the original query does not contain 'JOIN ON' keyword.



5. Move PROJECT operations down the query tree

Solution: