INFO20003 Tutorial – Week 8 Solutions

(Tutorial: Query optimisation)

Objectives:

This tutorial will cover:

I. Estimate cost of single-relation plans – 20 mins
II. Estimate cost of multi-relation plans – 35 mins

Exercises:

1. Single-relation plans:

Consider a relation with this schema:

Employees (*eid*: integer, *ename*: string, *sal*: integer, *title*: string, *age*: integer)

Suppose that the following indexes exist:

- An unclustered hash index on *eid*
- An unclustered B+ tree index on *sal*
- An unclustered hash index on *age*
- A clustered B+ tree index on (*age*, *sal*)

The Employees relation contains 10,000 pages and each page contains 20 tuples. Suppose there are 500 index pages for B+ tree indexes and 500 index pages for hash indexes. There are 40 distinct values of *age*, ranging from 20 to 60, in the relation. Similarly, *sal* ranges from 0 to 50,000 and there are up to 50,000 distinct values. *eid* is a candidate key; its value ranges from 1 to 200,000 and there are 200,000 distinct values.

For each of the following selection conditions, compute the Reduction Factor (selectivity) and the cost of the *cheapest* access path for retrieving all tuples from Employees that satisfy the condition:

a. sal > 20,000

The reduction factor (RF) is

$$\text{RF} = \frac{\text{High}(I) - \text{value}}{\text{High}(I) - \text{Low}(I)} = \frac{50{,}000 - 20{,}000}{50{,}000 - 0} = 0.6$$

There are two possible access paths for this query:

- The unclustered B+ tree index on *sal*, with cost

$$\begin{aligned}\text{Cost} &= \text{product of RFs of matching selects} \times (\text{NTuples}(R) + \text{NPages}(I)) \\ &= 0.6 \times \big((20 \times 10{,}000) + 500\big) \\ &= 120{,}300 \text{ I/Os}\end{aligned}$$

- Full table scan, with cost 10,000 I/Os.

Other indexes are not applicable here. Hence the cheapest access path is the full table scan, with cost 10,000.

b. age = 25

The reduction factor is

$$\text{RF} = \frac{1}{\text{NKeys}(I)} = \frac{1}{40}$$

Since we have two indexes on *age*, a hash index and a B+ tree index, there are three possible access paths for this query:

- The clustered B+ tree index on (*age*, *sal*), with cost

$$\begin{aligned}\text{Cost} &= \text{product of RFs of matching conditions} \times (\text{NPages}(R) + \text{NPages}(I)) \\ &= \frac{1}{40} \times (500 + 10{,}000) \\ &= 263 \text{ I/Os approx.}\end{aligned}$$

- The unclustered hash index on *age*, with cost

$$\begin{aligned}\text{Cost} &= \text{product of RFs of matching conditions} \times \text{hash lookup cost} \times \text{NTuples}(R) \\ &= \frac{1}{40} \times 2.2 \times (20 \times 10{,}000) \\ &= 11{,}000 \text{ I/Os}\end{aligned}$$

  For a hash index, the size does not matter as for each tuple the cost is 2.2; 1.2 is for the bucket check and 1 to fetch the page from the disk.

- Full table scan, with cost 10,000 I/Os.

Therefore, the cheapest access path here is to use the B+ tree index with cost 263 (approx.). Note that the full scan cost is the same as in the previous case.

c. age > 30

The reduction factor is

$$\text{RF} = \frac{\text{High}(I) - \text{value}}{\text{High}(I) - \text{Low}(I)} = \frac{60 - 30}{60 - 20} = 0.75$$

We cannot use the hash index over a range, thus the only options to consider are the full table scan vs. B+ tree index. There are two possible access paths for this query:

- The clustered B+ tree index on (*age*, *sal*), with cost

$$\begin{aligned}\text{Cost} &= \text{product of RFs of matching conditions} \times (\text{NPages}(R) + \text{NPages}(I)) \\ &= 0.75 \times (500 + 10{,}000) \\ &= 7875 \text{ I/Os}\end{aligned}$$

- Full table scan, with cost 10,000 I/Os.

Therefore, the clustered B+ tree index with cost 7875 is the cheapest access path here.

d. eid = 1000

As stated earlier, *eid* is a candidate key. Therefore, we can expect one record per *eid*. We can use the primary index (hash index on *eid*) to achieve a lookup cost of roughly

Cost = hash lookup cost + 1 data page access = 1.2 + 1 = 2.2

This is obviously cheaper than the full table scan (cost 10,000).

e. sal > 20,000 ∧ age > 30

The selection condition is the same as age > 30 ∧ sal > 20,000. This is similar to part c, so we can use the clustered B+ tree index, but the RF will be product of the RF for the two conditions, since both are applicable:

$$\mathrm{RF}_{age} = \frac{\mathrm{High}(I) - \mathrm{value}}{\mathrm{High}(I) - \mathrm{Low}(I)} = \frac{60 - 30}{60 - 20} = 0.75$$

$$\mathrm{RF}_{sal} = \frac{\mathrm{High}(I) - \mathrm{value}}{\mathrm{High}(I) - \mathrm{Low}(I)} = \frac{50{,}000 - 20{,}000}{50{,}000 - 0} = 0.6$$

There are two possible access paths for this query:

- The clustered B+ tree index on (*age*, *sal*), with cost

  Cost = product of RFs of matching conditions × (NPages(*R*) + NPages(*I*))
  = 0.75 × 0.6 × (500 + 10,000)
  = 4725 I/Os

- Full table scan, with cost 10,000 I/Os.

Thus the clustered B+ tree index on (*age*, *sal*), cost 4725, is the cheapest option here.

2. Multi-relation plans:

Consider the following schema:

Emp (<u>eid</u>, sal, age, $\overset{\text{FK}}{\text{did}}$)

Dept (<u>did</u>, $\overset{\text{FK}}{\underline{\text{projid}}}$, budget, status)

Proj (<u>projid</u>, code, report)

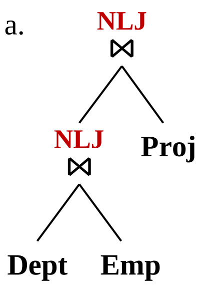
The number of tuples in Emp is 20,000 and each page can hold 20 records. The Dept relation has 5000 tuples and each page contains 40 records. There are 500 distinct *did*s in Dept. One page can fit 100 resulting tuples of Dept JOIN Emp. Similarly, Proj has 1000 tuples and each page can contain 10 tuples. Assuming that *projid* is the candidate key of Proj, there can be 1000 unique values for *projid*. The number of available buffer pages is 50, and Sort-Merge Join can be done in 2 passes. Let's assume that, if we join Proj with Dept, 50 resulting tuples will fit on a page.

Consider the following query:

```
SELECT E.eid, D.did, P.projid
FROM Emp AS E, Dept AS D, Proj AS P
WHERE E.did = D.did
  AND D.projid = P.projid;
```

For this query, estimate the cost of the following plans, focusing on the join order and join types:

a.



This left-deep plan is joining Dept with Emp using Nested Loop Join and then joining the results with Proj also using Nested Loop Join. The cost analysis is shown below:

Number of resulting tuples for Dept JOIN Emp

$$= \frac{1}{\text{NKeys}(I)} \times \text{NTuples(Dept)} \times \text{NTuples(Emp)}$$

$$= \frac{1}{500} \times 5000 \times 20{,}000$$

$= 200{,}000$ tuples

Number of pages for Dept JOIN Emp $= \frac{200{,}000}{100} = 2000$ pages

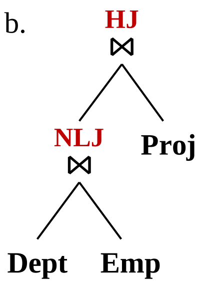
Cost of scanning Dept = 125 I/O

Cost to join with Emp = NPages(Dept) × NPages(Emp)
= 125 × 1000 = 125,000 I/O

Cost to join with Proj = NPages(Dept JOIN Emp) × NPages(Proj)
= 2000 × 100 = 200,000 I/O

Total cost = 125 + 125,000 + 200,000 = **325,125 I/O**

b.



This left-deep plan is joining Dept with Emp using Nested Loop Join and then joining the results with Proj using Hash Join. The cost analysis is shown below:

Number of resulting tuples for Dept JOIN Emp

$$= \frac{1}{\text{NKeys}(I)} \times \text{NTuples(Dept)} \times \text{NTuples(Emp)}$$

$$= \frac{1}{500} \times 5000 \times 20{,}000$$

$= 200{,}000$ tuples

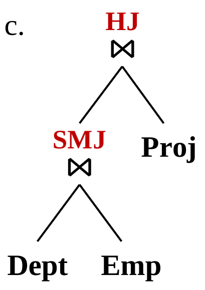
Number of pages for Dept JOIN Emp $= \frac{200{,}000}{100} = 2000$ pages

Cost of scanning Dept = 125 I/O

Cost to join with Emp = NPages(Dept) × NPages(Emp)
= 125 × 1000 = 125,000 I/O

Cost to join with Proj = 2 × NPages(Dept JOIN Emp) + 3 × NPages(Proj)
= 2 × 2000 + 3 × 100 = 4300 I/O

Total cost = 125 + 125,000 + 4300 = **129,425 I/O** c.



This left-deep plan is joining Dept with Emp using Sort-Merge Join and then joining the results with Proj using Hash Join. The number of passes of Sort-Merge Join is 2. The cost analysis is shown below:

Number of resulting tuples for Dept JOIN Emp

$$= \frac{1}{\text{NKeys}(I)} \times \text{NTuples(Dept)} \times \text{NTuples(Emp)}$$

$$= \frac{1}{500} \times 5000 \times 20{,}000$$

$$= 200{,}000 \text{ tuples}$$

Number of pages for Dept JOIN Emp $= \frac{200{,}000}{100} = 2000$ pages

Cost of sorting Dept = 2 × NPasses × NPages(Dept) = 2 × 2 × 125 = 500 I/O

Cost of sorting Emp = 2 × NPasses × NPages(Emp) = 2 × 2 × 1000 = 4000 I/O

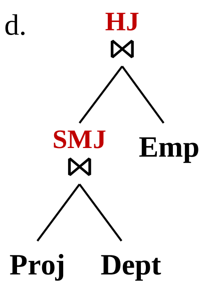
Cost of joining sorted Dept and Emp = NPages(Dept) + NPages(Emp)
= 125 + 1000 = 1125 I/O

Total cost of SMJ between Dept and Emp = 500 + 4000 + 1125 = 5625 I/O

Cost to join with Proj = 2 × NPages(Dept JOIN Emp) + 3 × NPages(Proj)
= 2 × 2000 + 3 × 100 = 4300 I/O

Total cost = 5625 + 4300 = **9925 I/O**

d.



This left-deep plan is joining Proj with Dept using Sort-Merge Join (with 2 passes) and then joining the results with Emp using Hash Join. The cost analysis is shown below:

Number of resulting tuples for Proj JOIN Dept

$$= \frac{1}{\text{NKeys}(I)} \times \text{NTuples(Proj)} \times \text{NTuples(Dept)}$$

$$= \frac{1}{1000} \times 1000 \times 5000$$

$$= 5000 \text{ tuples}$$

Number of pages for Proj JOIN Dept $= \frac{5000}{50} = 100$ pages

Cost of sorting Proj = 2 × NPasses × NPages(Proj) = 2 × 2 × 100 = 400 I/O

Cost of sorting Dept = 2 × NPasses × NPages(Dept) = 2 × 2 × 125 = 500 I/O

Cost of joining sorted Proj and Dept = NPages(Proj) + NPages(Dept)
= 100 + 125 = 225 I/O

Total cost of SMJ between Proj and Dept = 400 + 500 + 225 = 1125 I/O

Cost to join with Emp = 2 × NPages(Proj JOIN Dept) + 3 × NPages(Emp)
= 2 × 100 + 3 × 1000 = 3200 I/O

Total cost = 1125 + 3200 = **4325 I/O**