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

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

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

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

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

There are two selection conditions joined with "and". We calculate the RF for each condition:

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

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

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

Alternatively, we can use the unclustered B+ tree on *sal* and filter *age* on-the-fly afterwards. For this access path, the *age* condition does not match the index, so only the RF on *sal* will be used.

There are three possible access paths for this query:

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

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

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

$$\begin{aligned}\text{Cost} &= \text{product of RFs of matching conditions} \times (\text{NPages(R)} + \text{NPages(I)}) \\ &= 0.75 \times 0.6 \times (10{,}000 + 500) \\ &= 4725 \text{ I/Os}\end{aligned}$$

- 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>, <u>$\overset{\text{FK}}{\text{projid}}$</u>, budget, status)

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

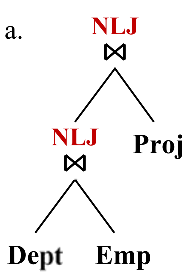
The number of tuples in Emp is 20,000 and each page can hold 20 records. The Dept relation has 5000 tuples and each page contains 40 records. There are 500 distinct *did*s in Dept. One page can fit 100 resulting tuples of Dept JOIN Emp. Similarly, Proj has 1000 tuples and each page can contain 10 tuples. Assuming that *projid* is the candidate key of Proj, there can be 1000 unique values for *projid*. 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. NLJ in this question means 'Page oriented NLJ'.

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

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

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

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

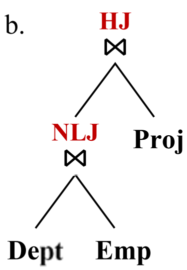
Number of pages for Dept JOIN Emp $= \frac{200{,}000}{100} = 2000$ pages

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

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

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

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

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

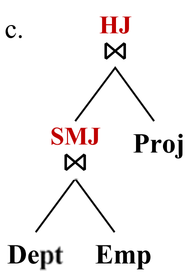
Cost of scanning Dept = 125 I/O

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

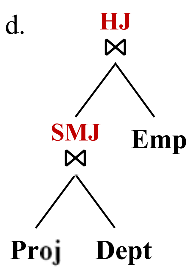
Cost of joining sorted Dept and Emp = NPages(Dept) + NPages(Emp)
= 125 + 1000 = 1125 I/O

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

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

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

$= 5000$ 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**

Take Home Questions:

1. **Multi Relation Plans with Access Methods:**

Consider the following schema:

Student (studentid, name, dob, degreename)

StudentSubject (studentid, subjectid, grade)

Subject (subjectid, name, level, coordinatorname, budget)

The number of tuples in Student is 20,000 and each page can hold 20 records. The StudentSubject relation has 50,000 tuples and each page contains 50 records. Subject has 1,000 tuples and each page can contain 10 records. One page can fit 100 resulting tuples of Student JOIN StudentSubject. 100 tuples resulting from the join of StudentSubject and Subject also fit onto a page. Assume that Subject.subjectid and Student.studentid are candidate keys. Sorting can be done in 2 passes.

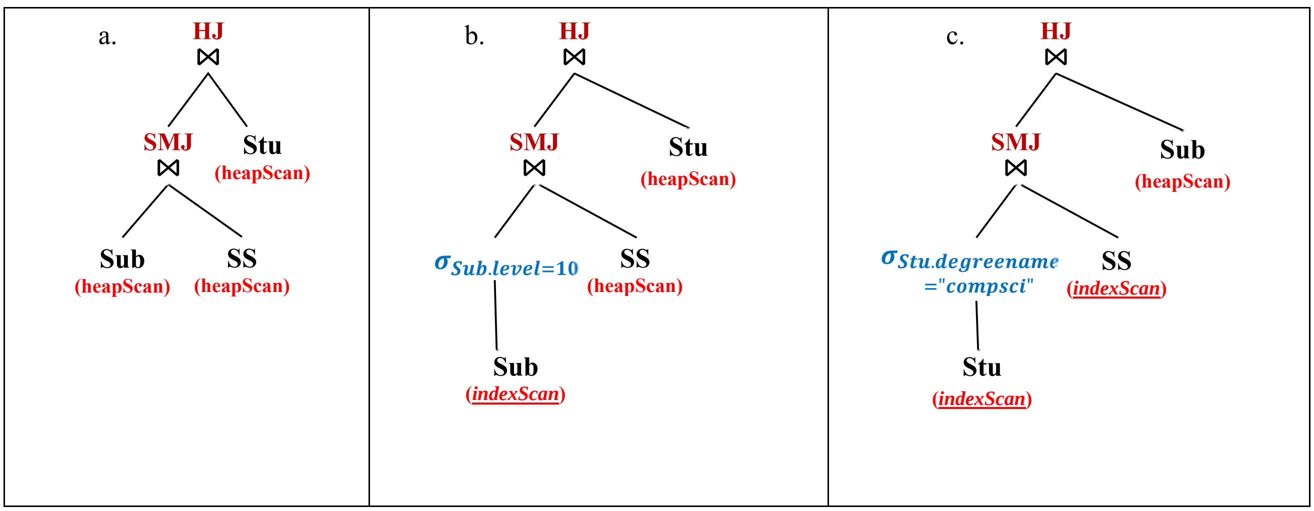
There are 3 available indexes: an unclustered hash index on Student(degreename), an unclustered B+ tree index on Subject(level), and a clustered B+ index on StudentSubject(studentid). All indexes have 50 pages.

There are 10 distinct values for Subject.level, ranging from 1-10. There are known to be 40 distinct values for Student.degreename.

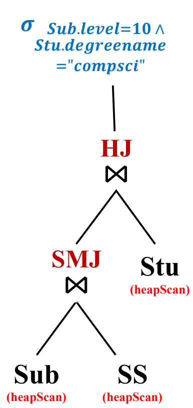
Consider the following query:

```
SELECT Stu.studentid, Sub.subjectid
FROM Student AS Stu, Subject AS Sub, StudentSubject AS SS
WHERE Stu.studentid = SS.studentid
  AND SS.subjectid = Sub.subjectid
  AND Stu.degreename = 'CompSci'
  AND Sub.level = 10
```

For this query, estimate the cost of the following plans. If selections are not marked on the tree, assume that they are done on the fly (in memory) **after** having completed all joins.



a. showing implicit Selections



This first plan is using only HeapScan for access, and selections are only performed on the fly (after all joins completed, see diagram to the left with the implicit selections)

Size of result for (Sub ⋈ SS):

$= \frac{1}{\text{NKeys(subjectid)}} \times \text{NTuples(Sub)} \times \text{NTuples(SS)}$

$= 1/1{,}000 \times 1{,}000 \times 50{,}000$

$= 50{,}000$ tuples

$= 50{,}000/100 = 500$ pages

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

Cost of sorting SS = 2 × NPasses × NPages(SS) = 2 × 2 × 1,000 = 4,000 I/O

[NOTE, we consider that 'heapscan' is not 'aware'/does not make use of the underlying sort of this relation, even though it is sorted because a clustered B+ index exists]

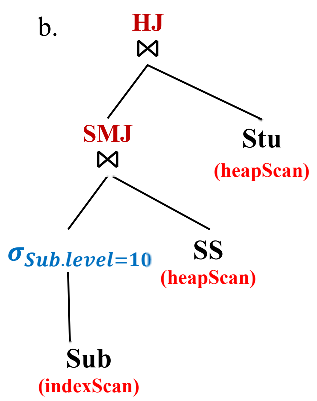
Cost of joining sorted Sub and SS = NPages(Sub) + NPages(SS)
= 100 + 1,000 = 1,100 I/O

Total cost of SMJ between Sub and SS = 400 + 4,000 + 1,100 = 5,500 I/O

Cost of HJ join with Stu:
= 2 × NPages(Sub ⋈ SS) + 3 × NPages(Stu) [due to pipelining]
= 2 × 500 + 3 × 1,000 = 4,000 I/O
Total cost = 5,500 + 4,000 = **9,500 I/O**

---

b.



This second plan uses an index to access the data instead of heapscan, and is also performing a selection along the way.

First, we calculate the RFs for selections

- Condition (Sub.level = 10) gives a RF of 1/10

Next, let's do the calculations involving the selections.

- First, selection for 'Sub.level =10' using unclustered B+ tree
  - Number of tuples selected
    - Ntuples(Sub) * RFs
    - = 1000*1/10
    - = 100 tuples = 10 pages
  - Cost of Selection
    - (Npages(index) + Ntuples(Sub))*RFs
    - = (50 + 1000) * 1/10
    - = 105 I/Os

Now, let's consider the child join: ($\sigma_{Sub.level=10}$ (Sub) ⋈ SS)

- First, calculate the size of the join result (the join key here is 'subjectId')
  - $= \dfrac{1}{\text{MAX(NKeys(subjectid))}} \times$ NTuples($\sigma_{Sub.level=10}$ (Sub)) × NTuples(SS)
  - = 1/1000 × 100 × 50,000
  - = 5,000 tuples
  - = 5,000/100 = 50 pages

- Next, cost the join
  - We're accessing SS using Heap Scan, so cost of accessing + sorting SS is =
    - 2*2*Npages(SS)
    - = 4000 I/Os
  - Now consider cost of sorting ($\sigma_{Sub.level=10}$ (Sub)) for SMJ
    - = 2 × NPasses × NPages($\sigma_{Sub.level=10}$ (Sub)) – NPages($\sigma_{Sub.level=10}$ (Sub)) [subtract Npages because of **pipelining:** we replaced the first read in of SMJ with the Index scan of Sub]
    - = 2 × 2 × 10 - 10 = 30 I/O

- Cost of merging sorted ($\sigma_{Sub.level=10}$ (Sub)) and SS
  - = NPages($\sigma_{Sub.level=10}$ (Sub)) + NPages(SS)
  - = 10 + 1000 = 1010 I/O
- Total cost of SMJ between ($\sigma_{Sub.level=10}$ (Sub)) and SS
  - = 30 + 4000 + 1010 = 5040 I/O

Great, let's now think about the parent join: (($\sigma_{Sub.level=10}$ (Sub) ⋈ SS) ⋈ Stu)

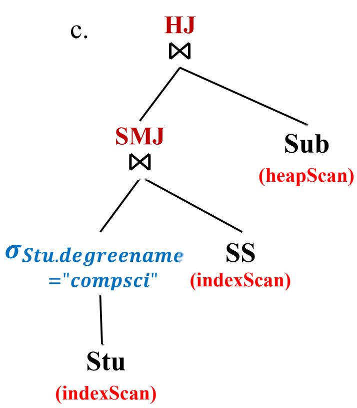
- Cost of the upper join:
  - = **2** × NPages(($\sigma_{Sub.level=10}$ (Sub)) ⋈ SS) + 3 × NPages(Stu) **[pipelining]**
  - = 2 × 50 + 3 × 1000
  - = 3100 I/O

So, the total cost is then:

- = 105 + 5040 + 3100
- = **8245 I/O**

---

c.



This third plan uses different indexes, and performs a different selection earlier in the plan evaluation.

First, we calculate the RFs for selections

- Condition (Stu.degreename = 'compsci') gives an RF of 1/40

Next, let's do the calculations involving the selections.

- First, selection for Stu.degreename = 'compsci' using unclustered hash index
  - Number of resulting tuples
    - Ntuples(Stu) * RFs
    - = 20,000*1/40
    - = 500 tuples = 25 pages
  - Cost of selection
    - 2.2*Ntuples(Stu)*RFs
    - = 2.2*20,000*1/40 = 1100 I/Os

Now, let's consider the child join: ($\sigma_{Stu.degreename = "compsci"}$ (Stu) ⋈ SS)

- First, calculate the size of the join result (the join key here is 'studentId')
  - $= \frac{1}{\text{MAX(NKeys(studentId))}}$ × NTuples($\sigma_{Stu.degreename = "compsci"}$ (Stu)) × NTuples(SS)
  - = 1/20,000 × 500 × 50,000
  - = 1,250 tuples
  - = 1,250/100 = 12.5 = 13 pages

- Now cost out the join

- We're accessing SS using an Index Scan instead of a Heap Scan, so we need to calculate the cost of first access. Note RF =1 since getting everything out, there is no selection criteria given.
    - = (Npages(index) + Npages(SS))*RFs
    - = (50+1000)*1
    - = 1050 I/Os
- Now, accessing & sorting ($\sigma_{\substack{Stu.degreename \\ ="compsci"}}(Stu)$) for SMJ:
    - = 2 × NPasses × NPages($\sigma_{\substack{Stu.degreename \\ ="compsci"}}(Stu)$) – NPages($\sigma_{\substack{Stu.degreename \\ ="compsci"}}(Stu)$) [subtract Npages because we replaced the first readin with the Index scan of Stu, so we **pipeline** from IndexScan to sorting]
    - = 2 × 2 × 25 - 25 = 75 I/O
- Cost of sorting SS
    - = 0 [sorted from the IndexScan already, since index is sorted!]
- Cost of merging sorted ($\sigma_{\substack{Stu.degreename \\ ="compsci"}}(Stu)$) and SS
    - = NPages($\sigma_{\substack{Stu.degreename \\ ="compsci"}}(Stu)$) + NPages(SS) – Npages(SS) [due to **pipelining** of SS from IndexScan]
    - = 25 + 1000 - 1000 = 25 I/O
- Total cost of SMJ:
    - = 75 + 0 + 25 = 100 I/O

Great, lets now think about the parent join:
($\sigma_{\substack{Stu.degreename \\ ="compsci"}}(Stu)$ ⋈ SS) ⋈ Sub)

- Cost of the parent join:
    - = 2 × NPages($\sigma_{\substack{Stu.degreename \\ ="compsci"}}(Stu)$ ⋈ SS) + 3 × NPages(Sub) [**pipelining**]
    - = 2 × 13 + 3 × 100
    - = 326 I/O

So, the total cost is then:

- = 1100+1050+100+326
- **= 2576 I/O**
    - [note how the estimated cost is significantly reduced compared to parts a and b!]