INFO20003 Tutorial 8

starting ~2.20 pm

Today's tutorial

- Cost Estimation of
  - single-relation plans
  - multi-relation plans
- Exercises
  - group work

It's really important that you try doing these questions yourself!

Exercise 1: Group work (10-15 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
b. age = 25
c. age > 30
d. eid = 1000
e. sal > 20,000 ∧ age > 30

1. **Reduction factor (Selectivity)**
   a. Col = value
      RF = 1/NKeys(Col)
   b. Col > value
      RF = (High(Col) – value) / (High(Col) – Low(Col))
   c. Col < value
      RF = (val – Low(Col)) / (High(Col) – Low(Col))
   d. Col_A = Col_B (for joins)
      RF = 1/ (Max (NKeys(Col_A), NKeys(Col_B)))
   e. In no information about NKeys or interval, use a "magic number" 1/10
      RF = 1/10

3. **Indexing Cost**
   a. B+-tree index
      i. Just a single tuple (selection over a primary key)
         Cost = Height(I) + 1
      ii. Clustered index (multiple tuples)
         Cost = (NPages(I)+NPages(R)) * Π RFᵢ
      iii. Unclustered (multiple tuples)
         Cost = (NPages(I)+NTuples(R)) * Π RFᵢ
   b. Hash Index
      i. Just a single tuple (selection over a primary key)
         Cost = 1.2 + 1 = 2.2
      ii. Clustered index (multiple tuples)
         Cost = (NPages (R)) * Π RFᵢ * 2.2
      iii. Unclustered index (multiple tuples)
         Cost = (NTuples(R)) * Π RFᵢ * 2.2

Query processing cost formulae

Legend

Symbol	Description
NKeys(Col)	The number of distinct values of column Col
High(Col)	The highest value of column Col
Low(Col)	The lowest value of column Col
NTuples(R)	The number of tuples of relation R
NPages(R)	The number of pages of relation R
NPages(I)	The number of pages of index I
Height(I)	The height of index I
Π RF$_i$	The product of all reduction factors
Π NTuples(R$_i$)	The product of the numbers of tuples of all relations taking part in a join
num_passes(R)	The number of passes for sorting relation R
PF	Projection factor (portion of all columns)

1. **Reduction factor (Selectivity)**
   a. Col = value
      RF = 1/NKeys(Col)
   b. Col > value
      RF = (High(Col) – value) / (High(Col) – Low(Col))
   c. Col < value
      RF = (val – Low(Col)) / (High(Col) – Low(Col))
   d. Col_A = Col_B (for joins)
      RF = 1/ (Max (NKeys(Col_A), NKeys(Col_B)))
   e. In no information about NKeys or interval, use a "magic number" 1/10
      RF = 1/10

a. sal > 20,000

The reduction factor (RF) is

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

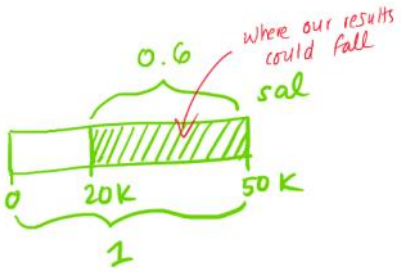
There are two possible access paths for this query:

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

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

- 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.

- How to calculate RF:
  - 

    - RF = 0.6 means that we expect/estimate only 0.6 (60%) of the relation will satisfy our condition (sal > 20,000)
    - RF can be in range [0,1]
    - Which one reduces cost more? A higher or lower RF?
    - What does a RF = 1 mean?

- Which formula to use:
  - b. Col > value
    RF = (High(Col) – value) / (High(Col) – Low(Col))
- Look through all the possible access paths
  - Only consider the paths which have an index on the relevant attributes
  - What other access path should we always calculate?

- ALWAYS calculate the cost of a heap scan
  - Heap Scan Cost = Npages(R)

- Solutions uses *I* - you should instead write the name of the column

2. Result size calculations

a. Single table

Result_size = NTuples(R) * Π RF_i

B-tree

iii. Unclustered (multiple tuples)

Cost = (NPages(I)+NTuples(R)) * Π RF_i

b. age = 25

The reduction factor is

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

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

  Cost = product of RFs of matching conditions × (NPages(*R*) + NPages(*I*))

  $= \frac{1}{40} \times (500 + 10{,}000)$

  = 263 I/Os approx.

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

  Cost = product of RFs of matching conditions × hash lookup cost × NTuples(*R*)

  $= \frac{1}{40} \times 2.2 \times (20 \times 10{,}000)$

  = 11,000 I/Os

  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.

- For Hash indexes, we don't care about the number of pages of the index - it's never used in any of the formulas
- We expect multiple tuples in our result (since age isn't a candidate key). This is why clustered B+ tree is better than Hash

B-tree

ii. Clustered index (multiple tuples)

- **Cost = (NPages(I)+NPages(R)) * Π RF_i**

Hash

iii. Unclustered index (multiple tuples)

Cost = (NTuples(R)) * Π RF_i * 2.2

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

  Cost = product of RFs of matching conditions × (NPages(*R*) + NPages(*I*))
  = 0.75 × (500 + 10,000)
  = 7875 I/Os

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

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

- Range query: shouldn't use hash index

B-tree on <age, sal>

ii. Clustered index (multiple tuples)

- **Cost = (NPages(I)+NPages(R)) * Π RFᵢ**

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).

b. Hash Index

i. Just a single tuple (selection over a primary key)

Cost = 1.2 + 1 = 2.2

e. sal > 20,000 ∧ age > 30

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

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

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

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

  Cost = product of RFs of **matching** conditions × (NTuples(R) + NPages(I))
  = 0.6 × ((20 × 10,000) + 500)
  = 120,300 I/Os (same as part a)

- 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 × (10,000 + 500)
  = 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.

- Be careful. Only can reduce RF if it is a **matching** condition!

B-tree on <sal>

iii. Unclustered (multiple tuples)

Cost = (NPages(I)+NTuples(R)) * Π RFᵢ

RF = RF(Sal)

B-tree on <age, sal>

ii. Clustered index (multiple tuples)

- **Cost = (NPages(I)+NPages(R)) * Π RF$_i$**

RF = RF(Age)*RF(Sal)

Exercise 2: Group work (20 mins)

2. Multi-relation plans:

Consider the following schema:

Emp (eid, sal, age, did) FK

Dept (did, projid, budget, status) FK

Proj (projid, code, report)

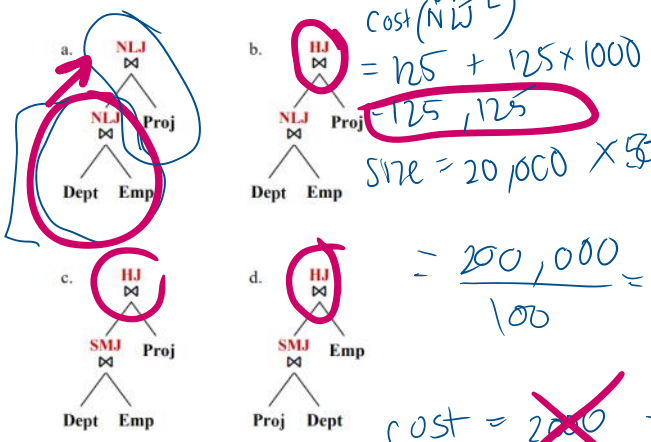
Emp	1000
Dept	125
Proj	100

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 *dids* 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:



Cost(NLJ E) = 125 + 125 x 1000 = 125,125

Size = 20,000 x 5000 x 1/500

= 200,000 / 100 = 2000 pg

cost = 2000 + 100 x 2000

325,125 I/O

ii. Page-oriented NLJ

Cost = NPages(R) + NPages (R) * NPages(S)

b. Hash Join

Cost = 3*(NPages(R) + NPages(S))

c. Sort-Merge Join

Cost$_{SMJ}$ = NPages(R) + NPages(S) +

2* NPages(R)* num_passes(R) +

2* NPages(S)* num_passes(S)

RF

d. Col_A = Col_B (for joins)

RF = 1/ (Max (NKeys(Col_A), NKeys(Col_B)))

Max (Did , Did)

2. Result size calculations

a. Single table

Result_size = NTuples(R) * Π RF$_i$

b. Joins

Result_size = Π NTuples(R) * Π RF$_i$

ii. Page-oriented NLJ

Cost = NPages(R) + NPages (R) * NPages(S)

b. Hash Join

Cost = 3*(NPages(R) + NPages(S))

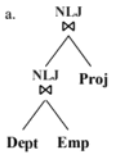
c. Sort-Merge Join

Cost$_{SMJ}$ = NPages(R) + NPages(S) +

2* NPages(R)* num_passes(R) +

2* NPages(S)* num_passes(S)

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

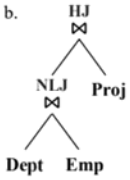
Cost to join with Emp = NPages(Dept) × NPages(Emp)
= 125 × 1000 = 125,000 I/O

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**

"Pipelined"
- discard the cost of the first read!
- only care about the cost of the join between the intermediate result and Proj

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

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

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

$= 200{,}000$ 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

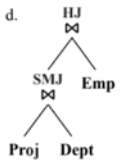
Cost of joining sorted Dept and Emp = NPages(Dept) + NPages(Emp)
= 125 + 1000 = 1125 I/O

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.

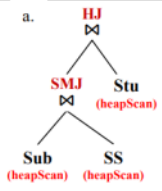
There are 10 distinct values for Subject.level, ranging from 1-10. There are known to be 40 distinct values for Student.degreename.

Consider the following query:

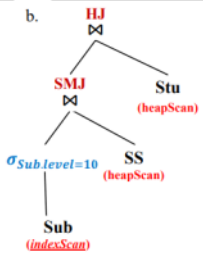
```
SELECT Stu.studentid, Sub.subjectid
FROM Student AS Stu, Subject AS Sub, StudentSubject AS SS
WHERE Stu.studentid = SS.studentid
  AND SS.subjectid = Sub.subjectid
  AND Stu.degreename = 'CompSci'
  AND Sub.level = 10
```

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.



b.



c.