03.03.2025 Week 3 Exercises: Query Optimization

Exercise 1

Consider the relations R(A, B), S(B, C), T(A, B, C).

1. Find a counter-example to show that:
   $\sigma_{A=\text{`a'}}((R \bowtie S) \cup T) \neq (\sigma_{A=\text{`a'}}(R) \bowtie S) \cup T$
2. Give the left hand side and the right hand side of the above expressions for your counter-example.
3. Modify the right hand side of the expression to make it equivalent to the left hand side.

Exercise 1 - Solution

1. R=(a,b), S = (b,c), T = (a1, b2, c3)
2. LHS = (a, b, c) ; RHS = (a,b,c), (a1,b2,c3)
3. $\sigma_{A='a'}((R \bowtie S) \cup T) \equiv (\sigma_{A='a'}(R) \bowtie S) \cup (\sigma_{A='a'}(T))$

Exercise 2

Consider the relation with schema EMP(ssn, name, age, jobcode) and the following statistics:

- There are n = 10,000 records in the file.
- There are 40 distinct values for *age* ranging from 21 to 60.
- There are 10 distinct values for *jobcode*.

Assume that all attributes (and index references) have the same size and you can fit 10 records per data page. There is an unclustered B-tree index on *age* and a clustered B-tree index on *jobcode*, both of height 2. Describe the most efficient way to execute each of the following queries and estimate the corresponding I/O cost:

1. `SELECT count(*) from EMP where jobcode = 'programmer'`
2. `SELECT count(*) from EMP where age > 40`
3. `SELECT count(*) from EMP where jobcode = 'programmer' and age > 40`

You can make any assumptions that you find necessary about the data distributions.

Exercise 2 - Solution

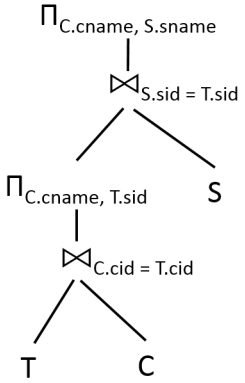
1. Using the index on jobcode, we can find and scan all the qualifying tuples. Assuming uniform distribution, there are (10,000 records / 10 jobcodes) = 1,000 programmers. We need 2 I/Os to reach the leaf level of the index. Then we need to scan the qualifying leaf pages. We do not need to access the data pages. Assume that each record in a leaf page is of the form (key, reference) and the reference has the same size as the key. Therefore, a leaf page can hold 10*2 = 20 records (twice as much as a data page). Cost: 2 + (1,000 records) / (20 records per leaf page) = 2 + 50 = 52 I/Os.

2. We assume that the records are uniformly distributed among the 40 age groups, thus half of the records qualify. We can use the index on age to find the qualifying tuples without accessing the data pages. The same reasoning as above applies, the only thing that changes is the selectivity. Cost: 2 + (5,000 records) / (20 records per leaf page) = 2 + 250 = 252 I/Os.

3. We can use the index on jobcode to find the first tuple that qualifies the first predicate. Then we access the data pages, exploiting the fact that the data is sorted on jobcode (because the index on jobcode is clustered). Therefore, having 1,000 tuples that qualify the first predicate, we need to access (1,000 records) / (10 records per data page) = 100 pages. Cost: 2 + 1 (for the leaf level) + (1,000 records) / (10 records per data page) = 3 + 100 = 103 I/Os.

Exercise 3

Consider the following schema:
Students S(sid, sname)
Taken T(sid, cid, grade, description)
Courses C(cid, cname)
Think of one way to improve the following query plan by applying a heuristic optimization rule.



Exercise 3 - Solution

Before the TC join, perform a projection for relation T to discard attributes *grade* and *description* that are not needed.

Exercise 4

Consider the following schema:
Students S(sid, sname)
Taken T(sid, cid, grade)
Courses C(cid, cname)

First Part

Consider first the query:

```
SELECT S.sname
FROM S, T, C
WHERE S.sid = T.sid and T.cid = C.cid
and  C.cname = 'CS101' and T.grade =  '6'
```

Estimate the time it takes to execute the above query for the two query plans show below. Consider only Block Nested Loop Join and Merge Join as the physical implementation of the join operators. The ids (*sid, cid*) are 4 bytes long, the names (*sname and cname*) are 60 bytes long and the *grade* is 4 bytes long. There are 33 buffer pages and the page size (excluding the header) is 1024 bytes. The Student relation contains 64,000 tuples, the Taken relation contains 128,000 tuples and the Courses relation contains 40 tuples. The relations S and T are ordered according to the sid column. There is a total of 1,000 grades of 6, 20 grades of 6 for the course CS101, and a total of 200 students taking the course CS101. It takes 0.1ms to transfer a page from disk to main memory. There are no indexes. Assume that the relations are stored on SSDs *i.e.*, they have no seek costs.

Provide the steps of your estimation. The number of buffers per operator are shown in brackets.

Second Part

Consider now the following query:

```
SELECT *
FROM S, T, C
WHERE S.sid = T.sid and T.cid = C.cid
```

Consider only Hash Join and Block Nested Loop Join as the physical implementation of the join operators. Assume the following join costs:

- HJ of S and T = 30
- BNL of S and T = 60
- HJ of C and T = 40
- BNL of C and T = 50
- HJ of the result of S⋈T and C = 40
- BNL of the result of S⋈T and C = 30
- HJ of the result of C⋈T and S = 50
- BNL of the result of C⋈T and S = 40

Do not consider plans that contain the join between C and S, as they will result in a Cartesian product. Depict how the System R query optimizer constructs *iteratively* the best query plan.

Exercise 4 - Solution

First Part

A naive solution is the following:

```
for each tuple s of S on disk do
for each tuple t of T on disk do
for each tuple c of C on disk do
if the two conditions on (s, t, c) hold then
output s.sname
```

The time cost of this naive solution is: $0.1 \cdot 64,000 \cdot 128,000 \cdot 40$ msec ≈ 1.04 years!
Below we analyze two reasonable solutions.

Solution-1

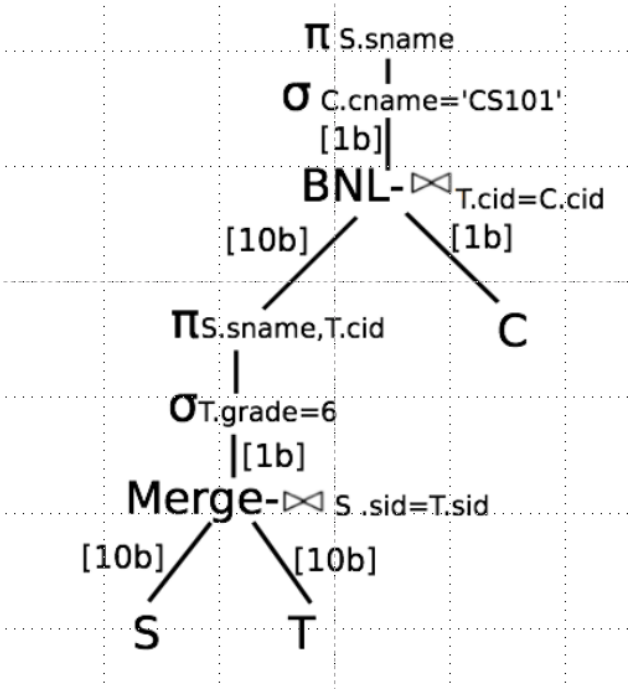


Figure 1: The 1st plan

Node	Tp size	#Tps/pg	#Tps	#Pgs	I/O pgs
S	60+4	16	64000	4000	-
T	4+4+4	85	128000	1506	-
1	76	13	128000	9847	4000+1506
2	76	13	1000	77	0
3	64	16	1000	63	0
C	60+4	16	40	3	-
4	128	8	1000	125	21
5	128	8	20	3	-
6	60	17	20	2	-
Total					5527

Table 1: Cost of the 1st plan

We now discuss the calculations below:

1. **(Nodes S and Node T)** Row for S and T are straightforward where we calculate the total number of pages required for storing the respective relations. We consider the I/O cost of reading them in the join operation as goes next.

2. **(Node 1 – Join)** First we estimate tuple size which is simply the sum of respective tuple sizes for relations S and T = 64 + 12 = 76. In this case, we assume that the

join attribute is duplicated in the output unless mentioned otherwise. This gives us 1024/76 = 13 tps/pg. The number of output tuples after the join will be the same as in the `Taken` relation, totaling to 128,000. To store these 128k tuples, we will need `ceil`(128k/13) pages = 9847.

(a) **I/O pgs:** We are given with the relations S and T sorted on the attribute `sid`. Hence, we consider only the cost of merging phase of the sort-merge join. The algorithm operates by maintaining two pointers – one for each relation. It then advances them one by one as depending on the relative value of attribute between the tuples of the two relations. This linear scan results in an I/O cost of #pgs of S + #pgs of T = 4000 + 1506 = 5506.

3. **(Node 2 – $\sigma_{\texttt{T.grade}}$)** This operator will not affect the resulting tuple size but only the number of output tuples. Since there are only 1000 grades of 6, only 1k out of 128k tuples will remain after the selection operator is applied. There is no I/O cost as the operator is applied to pages already in main memory.

4. **(Node 3 – $\pi_{\texttt{S.sname, T.cid}}$)** The projection operator retains only two attributes `(S.sname, T.cid)` which result in a tuple size of 64. Hence we can recalculate #tps/pg and total pages required to store the same 1k tuples as earlier.

5. **(Node C)** We count the number of pages for relation C using the tuple sizes. This is a straightforward calculation similar to Node S and Node T. Note again that we will include the I/O cost of reading relation C in the BNL join which follows next.

6. **(Node 4 – BNL Join)** Yet again, we start by estimating the tuple size which is the sum of tuples size for Node 3 and Node C = 64 + 64 = 128. The number of output tuples will be same as that of Node 3 which derives from the `Taken` relation. Joining with relation C will just add more attributes (e.g. `c.cname`) and not filter out any tuples generated from Node 3. Thus we have 1024/128 = 8 tps/pg. Hence, to store 1k tuples we need 1k/8 = 125 pages.

   (a) **I/O pgs:** The Block Nested Loop (BNL) Join algorithm has the following structure.

```
for each buffer block of outer-table
  for each buffer block of inner-table
    retain matching tuples if attributes match
```

   The outer-table here refers to the left hand side of the join. Therefore, the inner table will be read as many as times as the outer for loop iterates = $\frac{\text{\#pgs of outer-table}}{\text{\#buffer blocks for outer-table}}$ = $\frac{63}{10}$ = 7 (`ceil` applied). Hence, the total I/O cost should be = #pgs of outer-table + #pgs of inner-table * 7. Note however that the pages of the outer-table will already be in memory as a result of pipelined execution. In other words, the output of the $\pi_{\texttt{S.sname, T.cid}}$ will be stored directly in the 10 buffer pages allocated for BNL join. Hence, we do not count the I/O cost of reading the outer-table which gives us total I/O = #pgs of C * 7 = 3 * 7 = 21.

7. **(Node 2 –** $\sigma_{\texttt{C.cname = CS101}}$**)** As before, selection operator does not change tuple size but only the number of output tuples. From the information that there are only 20 grades of 6 for the course CS101, we can estimate that the number of output tuples after this selection is applied = 20. This requires only 3 pages when storing 8 tps/pg. There is no I/O cost as the selection is pipelined with the output of BNL join.

8. **(Node 3 –** $\pi_{\texttt{S.sname}}$**)** To store `S.sname`, we need only 60 bytes. Therefore we can store 1024/60 = 17 tps/pg, resulting in 2 pages in total to store 20 tuples. Once again there is no I/O cost as the projection is pipelined with the output of previous selection operator.

Time Cost: time_per_IO * IO = $0.1 \cdot 5527$ msec ≈ 0.5527 sec.

Solution-2

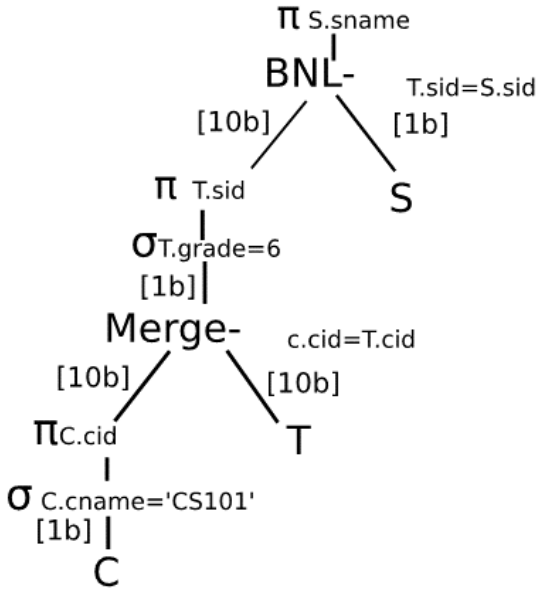


Figure 2: The 2nd plan

Node	Tp size	#Tps/pg	#Tps	#Pgs	I/O pgs
C	60+4	16	40	3	3
1,2	4	256	1	1	0
T	4+4+4	85	128000	1506	-
3	16	64	200	4	1506
4	16	64	20	1	0
5	4	256	20	1	0
S	60+4	16	64000	4000	-
6	68	15	20	2	4000
Total					5509

Table 2: Cost of the 2nd plan

The calculations for this solution are fairly similar to the previous example. We only discuss a few differences below:

1. **(Node C)** Unlike before, we include the I/O cost here since the relation must be read to process the σ and π before feeding to the join.
2. **(Node Merge-Join)**
   (a) **I/O pgs:** The linear scan of merge-join should result in an I/O cost of #pgs of outer-table + #pgs of inner-table. However, the outer-table is already in memory as a result of the projection operator. Therefore I/O cost = #pgs of T = 1506.

Time cost: $0.1 \cdot 5509$ msec ≈ 0.5509 sec.

Second Part

First do a HJ between S and T and then do a BNL between ST and C. The intermediate steps include dismissing BNL of S and T, BNL of C and T, HJ of ST and C, and HJ of the result of CT and S. Finally removing the below part; HJ of C and T as well as BNL of CT and S.