Exercise 8 : Query Plans

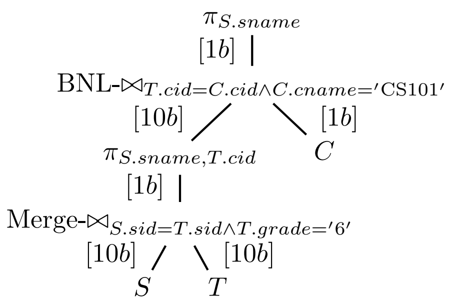
1. Consider the following environment and query:

   - Page size (excluding header): 1024 bytes
   - Schema: `Students S(sid,sname), Taken T(sid, cid, grade), Courses C(cid, cname)`
   - 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
   ```

   - Memory buffers: 33 pages
   - $\|S\|$ = 64000 tuples @ 4+60 bytes
     $\|T\|$ = 128000 tuples @ 4+4+4 bytes
     $\|C\|$ = 40 tuples @ 4+60 bytes
   - Assume a total of 1000 grades of 6, and 20 grades of 6 for course $CS101$, and that `S` and `T` are sorted on `sid` attribute

   Compute the number of I/O pages and the number of seeks it takes to compute the query using the execution plan below. The annotations of the form "$[nb]$" on the edges of the plan tree mean that n pages of the buffer are used to store the relation coming through that edge. Document your computation in the same way as was presented in the lecture - by providing a table with sizes and costs of the intermediate relations.



   Solution:

Node	tp size	#tps/pg	#tps	#pgs	I/O pgs	#seeks
S	64	16	64000	4000	4000	400
T	12	85	128000	1506	-	-
$S \bowtie C$	76	13	1000	77	1506	151
$\pi(ST)$	64	16	1000	63	0	0
C	64	16	40	3	-	-
$ST \bowtie C$	128	8	20	3	7 * 3	7 * 2
$\pi(STC)$	60	17	20	2	0	0
(total)					**5527**	**565**

   Total cost = 5527 * 0.1 + 565 * 10 ms = 6.2 s

2. Consider the following environment and query:

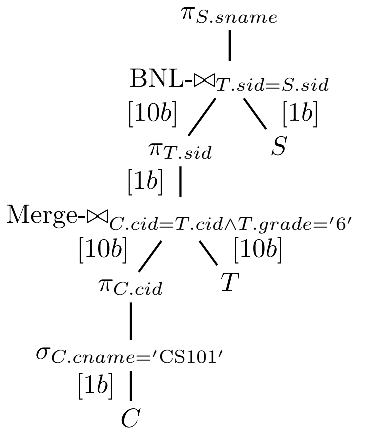
   - Page size (excluding header): 1024 bytes
   - Schema: `Students S(sid,sname), Taken T(sid, cid, grade), Courses C(cid, cname)`
   - Query:

   ```
   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}
   ```

   - Memory buffers: 33 pages

- $\|S\| = 64000$ tuples @ 4+60 bytes
  $\|T\| = 128000$ tuples @ 4+4+4 bytes
  $\|C\| = 40$ tuples @ 4+60 bytes
- Assume a total of 200 students taking course $CS101$ and 20 grades of 6 for this course, and that `C` and `T` are sorted on `cid` attribute

Compute the number of I/O pages and the number of seeks it takes to compute the query using the execution plan below. The annotations of the form "$[nb]$" on the edges of the plan tree mean that n pages of the buffer are used to store the relation coming through that edge. Document your computation in the same way as was presented in the lecture - by providing a table with sizes and costs of the intermediate relations.



Solution:

Node	tp size	#tps/pg	#tps	#pgs	I/O pgs	#seeks
C	64	16	40	3	3	1
$\pi(\sigma(C))$	4	256	1	1	0	0
T	12	85	128000	1506	-	-
$C \bowtie T$	16	64	20	1	1506	151
$\pi(CT)$	4	256	20	1	0	0
S	64	16	64000	4000	-	-
$CT \bowtie S$	68	15	20	2	1 * 4000	1
$\pi(CTS)$	60	17	20	2	0	0
(total)					**5509**	**153**

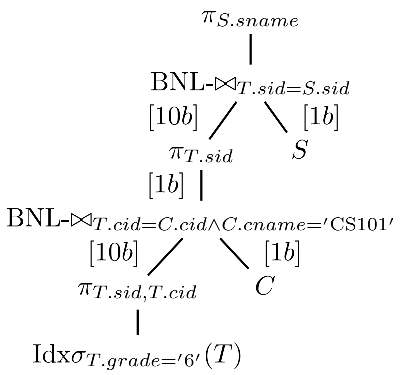
Total cost = 5509 * 0.1 ms + 153 * 10 ms = 2.08 s

3. Consider the following environment and query:
   - Page size (excluding header): 1024 bytes
   - Schema: `Students S(sid,sname), Taken T(sid, cid, grade), Courses C(cid, cname)`
   - 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}
```

   - Memory buffers: 23 pages
   - $\|S\| = 24000$ tuples @ 8+56 bytes
     $\|T\| = 384000$ tuples @ 8+8+4 bytes
     $\|C\| = 1600$ tuples @ 8+56 bytes
   - Assume: $1 : n$ relationship for all the joins, `cname` is a key (of course also `cid` and `sid` are keys), selectivity of grade = '6' is 15% (for every course), every course is taken by the same number of students, and that there is a clustered B-Tree index on `T.grade`

Compute the number of I/O pages and the number of seeks it takes to compute the query using the execution plan below. The annotations of the form "$[nb]$" on the edges of the plan tree mean that n pages of the buffer are used to store the relation coming through that edge. Document your computation in the same way as was presented in the lecture - by providing a table with sizes and costs of the intermediate relations.



Solution:

Node	tp size	#tps/pg	#tps	#pgs	I/O pgs	#seeks
Idx$\sigma(T)$	20	51	57600	1130	3 + 1130	3 + 1
$\pi(\sigma(T))$	16	64	57600	900	0	0
C	64	16	1600	100	-	-
$T \bowtie C$	80	13	36	3	90 * 100	90
$\pi(TC)$	8	128	36	1	0	0
S	64	16	24000	1500	-	-
$TC \bowtie S$	72	14	36	3	1 * 1500	1
$\pi(TCS)$	56	18	36	2	0	0
(total)					**11633**	**95**

Total cost = 11633 * 0.1 ms + 95 * 10 ms = 2.1 s

4. Consider the following environment and query:
   - Page size (excluding header): 1024 bytes
   - Memory buffers = 22 pages
   - Schema: `Students S(sid,sname,age,advisor_pid)`, `Take T(sid, cid)`, `Courses C(cid, cname,programme)`, `Professor P(pid, pname)`, `Teach E(pid, cid, semester)`
   - $\|S\||$ = 16000 tuples @ 4+60+4+4 bytes;
   - $\|T\|$ = 256000 tuples @ 4+4 bytes;
   - $\|C\|$ = 1600 tuples @ 4+60+4 bytes;
   - $\|P\|$ = 400 tuples @ 4+60 bytes;
   - $\|E\|$ = 32000 tuples @ 4+4+4 bytes

   The semester field identifies the year and semester (fall/spring) in which a course was taught by the professor. All students are undergraduate students. The course (degree) programme is 'computer science", "mathematics", "physics", etc. Assume that value distributions for fields such as age, programme, and semester are realistic for a place like EPFL.

   (a) There are no index structures. Provide an optimal query plan (identifying the join order and which operators to use) for the query template:

   ```
   select S.sid, E.pid from S, T, C, E
   where S.sid = T.sid and T.cid = C.cid and C.cid = E.cid
   and E.semester = <SOME GIVEN SEMESTER>
   and S.age = <SOME AGE>
   ```

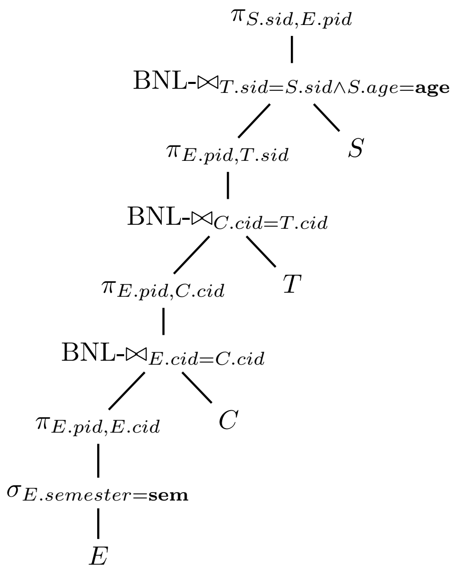
   Do NOT provide a cost calculation table, but use your insights about value distributions and data sizes to choose the query plan. Justify your decision: why this join order?

(b) Suppose you are allowed in total one and only one clustered B-tree index for the entire database. Which index (i.e., on which relation and which column(s)) would you choose to optimize execution speed of the above query template? Why? Would it change the optimal query plan, and if so, how?

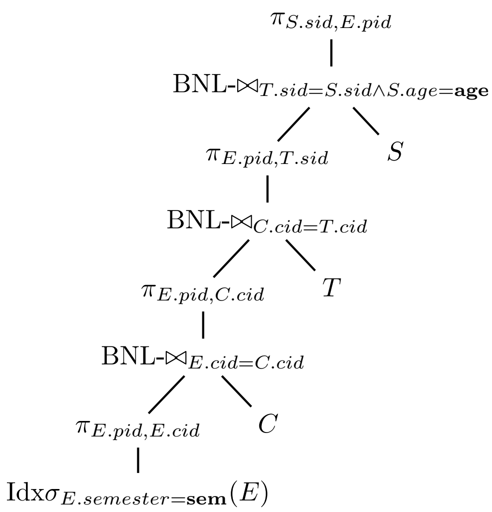
Solution:

(a) Since the semester field in E represents a year and semester identifier, we can assume it is more selective than the age attribute in S. For each of the joins, BNL join or hash join are reasonable. Below, after each join, project away all columns not needed anymore. The join with C can be removed by a smart query optimizer which has access to schema of the relations if the foreign key relation is explicitly specified. We assume that is not the case here and keep it. Description of an optimal query plan (bottom-up) :

i. Scan `E` and perform selection on the `semester` attribute and projection on the `pid` attribute.
ii. Do a pipelined join with `C`.
iii. Do a pipelined join with `T`.
iv. Do a pipelined join with `S`; include the selection on `S.age` in the join condition.



(b) Build a clustered index on `E.semester`, and replace step 1 of the query plan by and index scan on that index. We do this instead of replacing any of the Block-Nested Loop Join with Index-Nested Loop because the number of tuples are high compared to the number of pages and Index-Nested Loop Join is going to be worse in terms of performance.



5. Consider a directed graph given by a binary edge relation `E(from, to)`. Nodes are given by 64-bit unsigned integers. There is a clustered B-tree index on `E.from`. The graph has 1 billion edges; each node has on average 50 neighbors. Assume that the database system implements BNL-join and Index-NL-Join, but no other join algorithms. It implements all non-join operators we have discussed, such as projections, selections, index-selects,

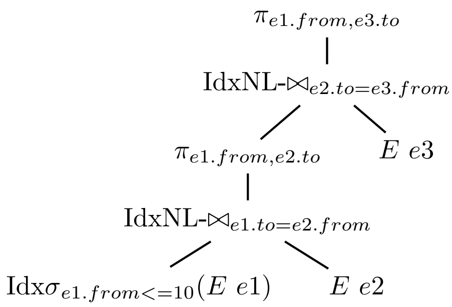
etc. The page size is 1024 bytes and there are 1000 memory buffer pages available. (That is, you may assume a very small number of extra pages if it simplifies the calculation.)

Find an optimal query plan for the query

```
select e1.from, e3.to
from   E e1, E e2, E e3
where  e1.to = e2.from
and    e2.to = e3.from
and    e1.from <= 10
```

and compute the cost of this plan. Show the query plan as an annotated operator tree. Show all calculation steps for (intermediate) costs and result sizes in a table as we did on our lecture slides. You do not need to execute the System R optimization algorithm but you must provide a convincing argument why the query plan you chose is optimal.

Solution:



Node	tp size	#tps/pg	#tps	#pgs	I/O pgs	#seeks
Idx$\sigma(Ee1)$	16	64	500	8	3+8	3+1
$Ee2$	16	64	10^9	15625000	-	-
$e1 \bowtie e2$	32	32	25000	782	500 * (3 + $\lceil \frac{50}{64} \rceil$)	500 * (3 + 1)
$\pi(e1e2)$	16	64	25000	391	0	0
$Ee3$	16	64	10^9	15625000	-	-
$e1e2 \bowtie e3$	32	32	1250000	39063	25000 * (3 + $\lceil \frac{50}{64} \rceil$)	25000 * (3 + 1)
$\pi(e1e2e3))$	16	64	1250000	78125	0	0
(total)					**102011**	**102004**

The join order is clear: we avoid Cartesian products and we do the filter on `e1` (using the index) first. Local cost calculation for the two operator alternatives shows that the choice of IdxNLJoin is right in both cases; `E` is simply huge.

$e1 \bowtie e2$:
IdxNLJ Cost: 500 * (3 + ceil(50/64)) = 2000 I/O, 2000 seeks, 2000*0.1+2000*10ms ≈ 20sec
BNLJ Cost: 1 * |E| = 15'625'000 I/O , 1seek, 1500sec

$e1e2 \bowtie e3$:
IdxNLJ Cost: 25000*(3+ceil(50/64))=100'000 I/O, 100'000 seeks = 100'000 * 10.1ms ≈ 1000sec
BNLJ Cost: 1 * $|E|$ = 15'625000 I/O, 1seek, 1500sec