Joins

We will be joining two tables: a table of students, and a table of assignment submissions; and we will be joining by the student ID:

CREATE TABLE Students (  
    student_id INTEGER PRIMARY KEY,  
    ...  
);  

CREATE TABLE AssignmentSubmissions(  
    assignment_number INTEGER,  
    student_id INTEGER REFERENCES Students(student_id),  
    ...  
);  

SELECT *  
FROM Students, AssignmentSubmissions  
WHERE Students.student_id = AssignmentSubmissions.student_id;

We also have:  
• Students has $S = 20$ pages, with $p_S = 200$ records per page  
• AssignmentSubmissions has $A = 40$ pages, with $p_A = 250$ records per page

Questions:  
1. What is the I/O cost of a simple nested loop join for Students $\bowtie$ AssignmentSubmissions?

2. What is the I/O cost of a simple nested loop join for AssignmentSubmissions $\bowtie$ Students?
3. What is the I/O cost of a block nested loop join for Students $\bowtie$ AssignmentSubmissions?
   Assume our buffer size is $B = 12$ pages.

4. What about block nested loop join for AssignmentSubmissions $\bowtie$ Students?
   Assume our buffer size is $B = 12$ pages.

5. What is the I/O cost of an Index-Nested Loop Join for Students $\bowtie$ AssignmentSubmissions?
   Assume we have a clustered alternative 2 index on AssignmentSubmissions.student_id, in the form of a height 2 B+ tree. Assume that index node and leaf pages are not cached; all hits are on the same leaf page; and all hits are also on the same data page.

6. Now assume we have a unclustered alternative 2 index on AssignmentSubmissions.student_id, in the form of a height 2 B+ tree. Assume that index node pages and leaf pages are never cached, and we only need to read the relevant leaf page once for each record of Students, and all hits are on the same leaf page.
   What is the I/O cost of an Index-Nested Loop Join for Students $\bowtie$ AssignmentSubmissions?
   HINT: The foreign key in AssignmentSubmissions may play a role in how many accesses we do per record.

7. What is the cost of an unoptimized sort-merge join for Students $\bowtie$ AssignmentSubmissions?
   Assume we have $B = 12$ buffer pages.
8. What is the cost of an optimized sort-merge join for Students × AssignmentSubmissions? Assume we have $B = 12$ buffer pages.

9. In the previous question, we had a buffer of $B = 12$ pages. If we shrunk $B$ enough, the answer we got might change.
   How small can the buffer $B$ be without changing the I/O cost answer we got?

10. What is the I/O cost of Grace Hash Join on these tables? Assume we have a buffer of $B = 6$ pages.

Query Optimization 1
(Modified from Fall 2017)

For the following question, assume the following:

- Column values are uniformly distributed and independent from one another
- Use System R defaults (1/10) when selectivity estimation is not possible
- Primary key IDs are sequential, starting from 1
- Our optimizer does not consider interesting orders

We have the following schema:

<table>
<thead>
<tr>
<th>Table Schema</th>
<th>Records</th>
<th>Pages</th>
<th>Indices</th>
</tr>
</thead>
</table>
| CREATE TABLE Student ( sid INTEGER PRIMARY KEY, name VARCHAR(32), major VARCHAR(64), semesters_completed INTEGER ) | 25,000 | 500 | • Index 1: Clustered(major). There are 130 unique majors  
• Index 2: Unclustered(semesters completed). There are 11 unique values in the range [0, 10] |
| CREATE TABLE Application ( sid INTEGER REFERENCES Student, cid INTEGER REFERENCES Company, status TEXT, (sid, cid) PRIMARY KEY ) | 100,000 | 10,000 | • Index 3: Clustered(cid, sid).  
• Given: status has 10 unique values |
| CREATE TABLE Company ( cid INTEGER PRIMARY KEY, open_roles INTEGER ) | 500 | 100 | • Index 4: Unclustered(cid)  
• Index 5: Clustered(open_roles). There are 500 unique values in the range [1, 500] |

Consider the following query:

```sql
SELECT Student.name, Company.open_roles, Application.referral
FROM Student, Application, Company
WHERE Student.sid = Application.sid  -- (Selectivity 1)
AND Application.cid = Company.cid  -- (Selectivity 2)
AND Student.semesters_completed > 6  -- (Selectivity 3)
AND (Student.major='EECS' OR Company.open_roles <= 50)  -- (Selectivity 4)
AND NOT Application.status = 'limbo'  -- (Selectivity 5)
ORDER BY Company.open_roles;
```

1. For the following questions, calculate the selectivity of each of the labeled Selectivities above.

   (a) Selectivity 1

   (b) Selectivity 2

   (c) Selectivity 3
2. For each predicate, which is the first pass of Selinger’s algorithm that uses its selectivity to estimate output size? (Pass 1, 2 or 3?)

(a) Selectivity 1
(b) Selectivity 2
(c) Selectivity 3
(d) Selectivity 4
(e) Selectivity 5

3. Mark the choices for all access plans that would be considered in pass 2 of the Selinger algorithm.

(a) Student ▹ Application (800 IOs)
(b) Application ▹ Student (750 IOs)
(c) Student ▹ Company (470 IOs)
(d) Company ▹ Student (525 IOs)
(e) Application ▹ Company (600 IOs)
(f) Company ▹ Application (575 IOs)

4. Which choices from the previous question for all access plans would be chosen at the end of pass 2 of the Selinger algorithm?

(a) Company ▹ (Application ▹ Student) (175,000 IOs)
(b) Company ▹ (Student ▹ Application) (150,000 IOs)
(c) Application ▹ (Company ▹ Student) (155,000 IOs)

5. Which plans that would be considered in pass 3?

(a) Company ▹ (Application ▹ Student) (175,000 IOs)
(b) Company ▹ (Student ▹ Application) (150,000 IOs)
(c) Application ▹ (Company ▹ Student) (155,000 IOs)
(d) Application $\bowtie$ (Company $\bowtie$ Student) (160,000 IOs)
(e) Student $\bowtie$ (Company $\bowtie$ Application) (215,000 IOs)
(f) (Company $\bowtie$ Application) $\bowtie$ Student (180,000 IOs)
(g) (Application $\bowtie$ Company) $\bowtie$ Student (200,000 IOs)
(h) (Application $\bowtie$ Student) $\bowtie$ Company (194,000 IOs)
(i) (Student $\bowtie$ Application) $\bowtie$ Company (195,000 IOs)
(j) (Student $\bowtie$ Company) $\bowtie$ Application (165,000 IOs)

6. Which choice from the previous question for all plans would be chosen at the end of pass 3?

Query Optimization 2
(Modified from Spring 2016)

1. True or False
   - When evaluating potential query plans, the set of left deep join plans are always guaranteed to contain the best plan.
   - As a heuristic, the System R optimizer avoids cross-products if possible.
   - A plan can result in an interesting order if it involves a sort-merge join.
   - The System R algorithm is greedy because for each pass, it only keeps the lowest cost plan for each combination of tables.

2. For the following parts assume the following:
   - The System R assumptions about uniformity and independence from lecture hold
   - Primary key IDs are sequential, starting from 1

   We have the following schema:
Consider the following query:

```sql
SELECT *
FROM Flight F, City C, Airline A
WHERE F.to_cid = C.cid
    AND F.aid = A.aid
    AND F.aid >= 2500
    AND C.population > 5e6
    AND C.state = 'California';
```

Considering each predicate in the WHERE clause separately, what is the selectivity for each?

(a) R1: `C.state='California'`

(b) R2: `F.to_id = C.cid`

(c) R3: `F.aid >= 2500`

(d) R4: `C.population > 5 * 10^6`

3. For each blank in the System R DP table for Pass 1. Assume this is before the optimizer discards any rows it isn’t interested in keeping and note that some blanks may be N/A. Additionally, assume B+ trees are height 2.
Detailed Solution:
(Note there is a typo in the exam’s solutions. population is *not* an interesting order for a the Index(III) scan.)

Flight:
Interesting order: aid is an interesting order because it’s used as part of a join condition in F.aid = A.aid, potentially making the algorithm choose an index nested loop join in later passes.

Cost: 2 + (100,020) * R3. First we have the R3 selectivity factor due to the F.aid ≤ 2500 clause and that this index is on F.aid. We traverse down to the leaves with 2 I/Os. Then we only have to read in part of the index that is relevant after applying the selectivity. The index size is 20 pages, but we read in R3 * 20 pages. Since the index is unclustered, we perform 1 I/O per matching tuple. We have 100K total tuples but only need to consider 100K * R3. This gives us a total of 2 + (100,000 + 20) * R3 I/Os for this index scan.

City 1st row:
Interesting order: None because file-scans don’t produce any interesting orders.

Cost: File-scans look through all of the pages, so it will take 20 I/Os.

City 2nd row:
Interesting order: None. population isn’t used in any later joins.

Cost: 2 + 30*R4. We have a selectivity factor of R4 since the index is on C.population. For clustered indexes, we perform 1 I/O per matching page of tuples. Therefore in a similar calculation to Flight, we read in R4*10pg portion of the relevant part of the index and R4*20pg worth of relevant pages of matching tuples.

4. After Pass 2, which of the following plans could be in the DP table?

(a) City [Index(III)] JOIN Airline [File scan]
(b) City [Index (III)] JOIN Flight [Index (I)]
(c) Flight [Index (II)] JOIN City [Index (III)]

Solution:
(a) Cannot. Because there is no condition joining City and Airline, this is a cross product, which the Selinger algorithm avoids.
(b) Can. City [Index (III)] is kept from pass 1 because it has the lowest cost of the cities table. Flight [Index (I)] is kept from pass 1 because it has an interesting order.
(c) Cannot. Index (II) would not have been kept as a Single Table Access Method. No interesting order and more expensive than a simple full scan.

5. Suppose we want to optimize for queries similar to the query above in part 2, which of the following suggestions could reduce I/O cost?

(a) Change Index (III) to be unclustered
(b) Store City as a sorted file on population
Solution:

(a) Won’t reduce I/O cost. An unclustered index would not minimize I/O cost, since it’s more random I/O, and we may load a page more than once. Instead of 1 I/O per matching page of tuples, this would increase the cost to 1 I/O per matching tuple.

(b) May reduce I/O cost. Sorted file may provide more efficient range lookups due to the presence of the \( C.\text{population} > 5e6 \) clause.