CS W186 - Spring 2020
Guerrilla Section 3
Joins and Query Optimization

Sunday, March 15, 2020

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`?

   Answer: **160,020 I/Os.**
   The formula for a simple nested loop join is $[S] + |S| \cdot [A]$.
   Plugging in the numbers gives us $20 + (20 \cdot 200) \cdot 40 = 160,020$ I/Os.

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

   Answer: **200,040 I/Os.**
   The formula for a simple nested loop join is $[A] + |A| \cdot [S]$.
   Plugging in the numbers gives us $40 + (40 \cdot 250) \cdot 20 = 200,040$ I/Os.

3. What is the I/O cost of a block nested loop join for `Students` ⋈ `AssignmentSubmissions`?

   Assume our buffer size is $B = 12$ pages.

   Answer: **100 I/Os.**
   First, we can calculate our block size: $B - 2 = 10$.
   Since `Students` is our left table, we calculate the number of blocks of `Students`:
   $[S]/(B-2) = 20/10 = 2$.
   Thus the final cost is $[S]$ plus 2 passes through all of $[A]$, or $20 + 2 \cdot 40 = 100$ I/Os.

4. What about block nested loop join for `AssignmentSubmissions` ⋈ `Students`?

   Assume our buffer size is $B = 12$ pages.

   Answer: **120 I/Os.**
   As before, we can calculate our block size: $B - 2 = 10$.
   Since `AssignmentSubmissions` is our left table, we calculate the number of blocks:
   $[A]/(B-2) = 40/10 = 4$.
   Thus the final cost is $[A]$ plus 4 passes through all of $[S]$, or $40 + 4 \cdot 20 = 120$ I/Os.

5. What is the I/O cost of an Index-Nested Loop Join for `Students` ⋈ `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.

   Answer: **16,020 I/Os.**
   The formula is $[S] + |S| \cdot$ ⟨cost of index lookup⟩.
   The cost of index lookup is 3 I/Os to access the leaf, and 1 I/O to access the data page for all matching records.
   So the total cost is $20 + 4000 \cdot 4 = 16,020$ I/Os.

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` ⋈ `AssignmentSubmissions`?

   HINT: The foreign key in `AssignmentSubmissions` may play a role in how many accesses we do per record.

   Answer: **22,020 I/Os.**
   The formula is $[S] + |S| \cdot \langle \text{cost of index lookup} \rangle$.
   This time though, the cost of index lookup is 3 I/Os to access the leaf, and 1 I/O to access the data page **for each matching record**.
   How many records match per key? We actually haven't told you! But, we do know that we will eventually have to access each record exactly once (since each `AssignmentSubmission` is foreign-keyed on a `student_id`) - so there will be $|A| = 10,000$ data page lookups, one for each row.
   So the total cost is $20 + 4000 \cdot 3 + 10000 = 22,020$ I/Os.

7. What is the cost of an unoptimized sort-merge join for `Students` ⋈ `AssignmentSubmissions`?

   Assume we have $B = 12$ buffer pages.

   Answer: **300 I/Os.**
   The formula is $\langle \text{cost of sorting S} \rangle + \langle \text{cost of sorting A} \rangle + [S] + [A]$.
   For sorting S: The first pass will make two runs, which is mergeable in one merge pass; thus, we need two passes.
   For sorting A: The first pass will make four runs, which is mergeable in one merge pass; thus, we need

two passes.
Thus the total cost is $(2 \cdot 2[S]) + (2 \cdot 2[A]) + [S] + [A] = 5([S] + [A]) = 5 \cdot 60 = 300$ I/Os.

8. What is the cost of an **optimized** sort-merge join for `Students` $\bowtie$ `AssignmentSubmissions`?

   Assume we have $B = 12$ buffer pages.

   Answer: **180 I/Os.**
   The difference from the above question is that we will skip the last write in the external sorting phase, and the initial read in the sort-merge phase.
   For this to be possible, all the runs of S and A in the last phase of external sorting should be able to fit into memory together. From the previous question, we know there are $2 + 4 = 6$ runs, which fits just fine in our buffer of 12 pages.
   Thus the total cost is $300 - 2[S] - 2[A] = 300 - 120 = 180$ I/Os.

9. In the previous question, we had a buffer of $B = 12$ pages. If we shrank B enough, the answer we got might change.

   How small can the buffer B be without changing the I/O cost answer we got?

   Answer: **9 buffer pages.**
   The restriction for optimized sort-merge join is that the number of final runs of S and A can both fit in memory simultaneously. (i.e., the number of runs of S + the number of runs of A $\leq$ B - 1). We had $2 + 4$ runs last time, which fit comfortably in $12 - 1$ buffer pages (recall that one page is reserved for output).
   What about $B = 11$? We would still have $2 + 4 < 11 - 1$ runs.
   What about $B = 10$? We would still have $2 + 4 < 10 - 1$ runs.
   What about $B = 9$? Now we have 3 runs for S and 5 runs for A, which just exactly fits in $9 - 1$ buffer pages.
   Since 9 buffer pages fits perfectly, any smaller would force more merge passes and thus more I/Os.

10. What is the I/O cost of Grace Hash Join on these tables?

    Assume we have a buffer of $B = 6$ pages.

    Answer: **180 I/Os**
    For Grace Hash Join, we have to walk through what the partition sizes are like for each phase, one phase at a time.
    In the partitioning phase, we will proceed as in external hashing. We will load in 1 page a time and hash it into $B - 1 = 5$ partitions.
    This means the 20 pages of S get split into 4 pages per partition, and the 40 pages of A get split into 8 pages per partition.
    Do we need to recursively partition? No! Remember that the stopping condition is that any table's partition fits in $B - 2 = 4$ buffer pages; the partitions of S satisfy this.
    In the hash joining phase, the I/O cost is simply the total number of pages across all partitions - we read all of these in exactly once.
    Thus the final I/O cost is $20 + 20$ for partitioning S, $40 + 40$ for partitioning A, and $20 + 40$ for the hash join, for a total cost of 180 I/Os.

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 Schema	Records	Pages	Indices
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:

```
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
   1/max(25000, 25000) = 1/25000. There are exactly 25000 values in Student.sid, and due to the foreign key, there are at most 25000 values of Application.sid.

   (b) Selectivity 2
   1/max(500, 500) = 1/500. Similarly to Selectivity 1, there are exactly 500 values in Company.cid, and due to the foreign key, there are at most 500 values in Application.cid.

   (c) Selectivity 3
   (10 - 6) / (10 - 0 + 1) = 4/11. We have 11 unique values, assumed to be equally distributed. Therefore we use the equation for less than or equal to which is (high key - value) / (high key - low key + 1).

(d) Selectivity 4
(1/130 + 1/10) - (1/130 * 1/10) = 10/1300 + 130/1300 - 1/1300 = 139/1300. We can find the selectivity that they are an EECS major by using the equation 1/distinct values. Next, we find the selectivity that open positions are less than or equal to 50 using the equation (v - low key) / ((high key - low key + 1) + (1 / number distinct)). Lastly we combine these two selectivities using S(p1) + S(p2) - S(p1)S(p2) to determine the selectivity of having one or the other.

(e) Selectivity 5
1 - (1/10) = 9/10. Given 10 unique values, the non-negated predicate has selectivity 1/10, so we can use the equation for NOT which is 1 - selectivity of the predicate. The selectivity of the predicate is 1/10 (because there are 10 unique values).

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

   Solution: Pass 2, Pass 2, Pass 1, Pass 3, Pass 1. C and E are pass 1 because they only involve filtering one table. A and B are pass 2 because they represent a join. Note that (d)—the OR predicate—is over 2 tables that have no associated join predicate, so the selection is postponed along with the cross-product, until after 3-way joins are done.

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

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

   A, B, E, and F will be considered because they are not cross products.They are joined on a condition, so some rows can be filtered out, making our intermediate relations smaller.

4. Which choices from the previous question for all access plans would be chosen at the end of pass 2 of the Selinger algorithm?
   B and F will be chosen because they have the lower cost for joining the two tables, and we have the assumption that our optimizer does not consider interesting orders. (Even if we did, there are no interesting orders in the other joins.)

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

   (a) Company $\bowtie$ (Application $\bowtie$ Student) (175,000 IOs)
   (b) Company $\bowtie$ (Student $\bowtie$ Application) (150,000 IOs)
   (c) Application $\bowtie$ (Company $\bowtie$ 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)

Considers F and H only. A-E can be immediately discarded because they aren't left-deep. G won't be considered because we chose (Company $\bowtie$ Application) in pass 2. Similarly, choice I wouldn't be considered because we choose Application $\bowtie$ Student in the previous pass. Choice J wouldn't be considered because there is no join condition on Student and Company, so this is a cross-join, which we avoid since we have other options.

6. Which choice from the previous question for all plans would be chosen at the end of pass 3?
Chooses F. F has the lower I/O cost between F and H.

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.

   False. This is a heuristic that System R uses to shrink the search space.
   True.
   True. Sort merge join leaves the joined tables in sort order, which may be useful in future passes and/or if the overall query includes an ORDER BY clause.
   False. It is not greedy because it keeps track of interesting orders. (Dynamic Programming!)

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:

`CREATE TABLE Flight (`
`fid INTEGER PRIMARY KEY,`
`from_id INTEGER REFERENCES City,`
`to_id INTEGER REFERENCES City,`
`aid INTEGER REFERENCES Airline)`
	NTuples: 100K, NPages: 50
	Index:
	(I) unclustered B+-tree on aid. 20 leaf pages.
	(II) clustered B+-tree on (from_cid, fid). 10 leaf pages.

`CREATE TABLE City (`
`cid INTEGER PRIMARY KEY,`
`name VARCHAR(16),`
`state VARCHAR(16),`
`population INTEGER)`
	NTuples: 50K, NPages: 20
	Index:
	(III) clustered B+-tree on population. 10 leaf pages. (IV) unclustered index on cid. 5 leaf pages. Statistics:
	state in [1, 50], population in [10^6, $8*10^6$]

`CREATE TABLE Airline (`
`aid INTEGER PRIMARY KEY,`
`hq_cid INTEGER REFERENCES City,`
`name VARCHAR(16))`
	NTuples: 5K, NPages: 2

Consider the following query:

```
SELECT *
FROM Flight F, City C, Airline A
WHERE F.to_id = 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'`

1/50, since there are 50 possible values for state.

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

1/MAX(50000, 50000) = 1/50000, since there are 50000 tuples in the City table, and cid is the primary key of the City table, while to_id is a foreign key that references the primary key of City, so there are also 50000 values for it.

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

Since we have the assumption that primary key IDs are sequential, starting from 1, we know that the low value for F.aid (which is a primary key) is 1, and the high value is 5000 (since we have 5000 records). Thus, our selectivity is (5000 - 2500)/(5000 - 1 + 1) + 1/5000 = 2501/5000.

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

$$\frac{(8*10^6)-(5*10^6)}{(8*10^6)-(1*10^6)+1} = \frac{3*10^6}{(7*10^6)+1}$$

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 it takes 2 I/Os to reach the leaf nodes.

Table(s)	Plans	Interesting Orders from Plan (N/A if none)	Cost (I/Os)
Flight	Index (I)	aid	2 + 100020*R3
City	Filescan	N/A	20
City	Index (III)	N/A	2 + 30*R4

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 $\leq$ 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.population > 5e6 clause.