CS 186 - Fall 2020
Exam Prep Section 3
Joins and Query Optimization

Sunday, October 18, 2020

Joins 1

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` $\bowtie$ `AssignmentSubmissions`?
   Assume we have $B = 12$ buffer pages.

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?

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

Joins 2

Consider a modified version of the baseball database that only stores information from the last 40 years.

```
CREATE TABLE Teams (
    team_id INTEGER PRIMARY KEY,
    team_name VARCHAR(20),
    year INTEGER,
    ...
);

CREATE TABLE Players(
    player_id INTEGER PRIMARY KEY,
    team_id INTEGER REFERENCES Teams(team_id),
    year INTEGER,
    ...
);
```

Each record in Teams represents a single team for a single year. Each record in Players represents a single player during a single year. We also have:

- `Teams` has $[T] = 30$ pages, with $p_T = 40$ records per page
- `Players` has $[P] = 300$ pages, with $p_P = 50$ records per page

Questions:

1. What is the I/O cost of a simple nested loop join for joining `Teams` $\bowtie$ `Players` on `Teams.team_id` = `Players.team_id`?

2. What is the I/O cost of a page nested loop join on the same query?

3. What is the I/O cost of a block nested loop join on the same query? Assume our buffer size is $B = 10$ pages.

4. Assume we have an unclustered index of height 1 on `Teams.team_id`. What is the I/O cost of an index nested loop join on `Teams.team_id` = `Players.team_id` using this index? You can assume that every player only plays on one team each year.

5. Now, assume we have a clustered index of height 2 on `Players.player_id` and an clustered index of height 3 on `Players.team_id`. If each team has 25 players each year, what is the lowest I/O cost of the join on on `Teams.team_id` = `Players.team_id` using one of these indexes?

6. Assume the index on `Players.team_id` is actually unclustered. What is cost of the join on on `Teams.team_id` = `Players.team_id` using this index?

7. Consider a universe where there are no limits on team size and players get to time travel to play for whatever team they want, and 75% of players choose to travel to 2020 to play for the best baseball team, the San Diego Padres. **In other words**, imagine that 75% of the records in the `Players` table have the same team_id. What are the effects on the performance of the different join algorithms? *Hint: Consider which of the joins are affected by duplicates. Remember that only one of the tables has duplicates, while the other does not.*

Joins 3

In the following problems, we will be joining two tables: Students and AssignmentSubmissions on the key 'student_id'. However, we are dealing with a set of system constraints. Given a set of potential join algorithms from SNLJ, BNLJ, PNLJ, Hash Join, GHJ, SMJ, select the best option(s).

```
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;
```

Unless otherwise explicitly stated, there are 600 pages of Students, 600 pages of AssignmentSubmissions, 60 records/page for each and 10 buffer pages. Assume all student_ids are unique between both tables. All parts are independent of each other.

1. Our program memory is extremely limited (not buffer memory)! As a result, we only have enough memory to store 1 hash function. What join algorithms work here provided it fits our system constraints)? Why?

2. Now, we have very little buffer memory (only 3 pages). What join algorithms work here? How are the different join algorithms affected by the given constraint? Why?

3. Now, assume that Students is only 1 page. What is the best join algorithm IO wise?

4. Now, assume that all student_ids in both tables are exactly the same (i.e. assume the primary key constraint does not hold). What join algorithms work here? How are the different join algorithms affected by the given constraint? Why?

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

(b) Selectivity 2

(c) Selectivity 3

(d) Selectivity 4

(e) Selectivity 5

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 $\bowtie$ 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)

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

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)

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:

`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_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.

Table(s)	Plans	Interesting Orders from Plan (N/A if none)	Cost (I/Os)
Flight	Index (I)		
City	Filescan		
City	Index (III)		

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)]

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