COSC 460 Lecture 13: Query Processing 3: Joins

Professor Michael Hay

Fall 2018

Join algorithms

- Nested loops: simple, page, block, index
- Sort-merge
- Hash

Notation and Example

```
select *
From R, S where R.A = S.B
```

- N_R = number of pages in R
- p_R = number of tuples/page in R
- M = memory (number of frames in buffer pool)
- Example:
  - $N_R = 1000$, $p_R = 100$
  - $N_S = 500$, $p_R = 80$
  - $M = 102$
  - Assume page I/O is 10ms

(Simple) nested loops

```
for each t in R:
   for each t' in S:
      if t and t' match:
         add join(t,t') to result
```

Cost analysis:

$N_R + (p_R \times N_R) \times N_S$

Cost (on example): when R is outer relation

$1000 + (100 \times 1000) \times 500 = 50{,}001{,}000$ I/Os ≈ **140 hours**

Poll

Instructions: *I will give you 1-2 minutes to think on your own.*
Vote 1.
Then you will discuss w/ neighbor (1 min).
Vote 2.
Then we'll discuss as class.

Correct answer: E

Simple nested loops only *requires* 3 pages of memory (one for R, one for S, one for output). Now suppose we increase buffer pool to be of size M, for some M > 3. **How does a larger buffer pool affect the cost of nested loops?** *You can assume that the eviction policy is LRU and the pages of the "current" tuples are pinned while the tuple is in use.*

A. Cost is increased

B. Cost is unchanged

C. Cost is reduced

D. Cost is reduced when $M \geq 2 + N_R$.

E. Cost is reduced when $M \geq 2 + N_S$.

Cost analysis of simple nested loops:

$$N_R + (p_R \times N_R) \times N_S$$

Page nested loops

```
for each page p in R:
    for each page p' in S:
        for each t in p:
            for each t' in p':
                if t and t' match:
                    add join(t,t') to result
```

Cost analysis:

$N_R + N_R \times N_S$

Cost (on example): when R is outer relation

$1000 + 1000 \times 500 = 501{,}000$ I/Os ≈ **1.4 hours**

Poll

Instructions: *I will give you 1-2 minutes to think on your own.*
Vote 1.
Then you will discuss w/ neighbor (1 min).
Vote 2.
Then we'll discuss as class.

Correct answer: B

How many pages of memory does page nested loops require?

A. $M = 2$

B. $M = 3$

C. $M = \min(N_R, N_S)$

D. $M = \max(N_R, N_S)$

E. $M = N_R + N_S$

Cost analysis of page nested loops:
$N_R + N_R \times N_S$

Exercise

Instructions: discuss with neighbors.

Briefly discuss whether/how you could implement page-nested loops in ColgateDB.

Block nested loops

```
for each [block of M-2 pages] in R:
  for each page p' in S:
    for each t in block:
      for each t' in p':
        if t and t' match:
          add join(t,t') to result
```

Cost analysis:
N_R + **ceiling**(N_R/(M-2)) × N_S

Cost (on example): when R is outer relation
1000 + 1000/(102-2) × 500 = 6,000 I/Os ≈ **1 minute**

Poll

Instructions: *I will give you 1-2 minutes to think on your own.*
Vote 1.
Then you will discuss w/ neighbor (1 min).
Vote 2.
Then we'll discuss as class.

Correct answer: D

Consider this variant of block-nested loops: instead of taking a block of the *outer* relation, we take a block of the *inner* relation. How does this change affect cost? (Assume LRU eviction.)

```
for each each page p in R:
  for [block of M-2 pages] in S:
    for each t in block:
      for each t' in p:
        if t and t' match:
          add join(t,t') to result
```

A. Cost is same

B. Cost is lower

C. Cost is higher

D. Cost depends on the size of M relative to size of S

Hash join

Hash function should partition records into M-1 "buckets" (aka partitions). *Why M-1?*

```
Hash R on A: write results to P1(R)…PM-1(R)
Hash S on B: write results to P1(S)…PM-1(S)
Join partitions Pi(R) and Pi(S) for all i
```

Cost analysis:

$2 \times N_R +$
$2 \times N_S +$
$N_R + N_S$ *(best case: assumes for each pair, $P_i(R)$ and $P_i(S)$, at least one partition fits in memory)*

Cost (on example):

$3 \times (500 + 1000) = 4{,}500$ I/Os ≈ **45 seconds**

Exercise

Instructions: discuss with neighbors.

Suppose $N_R > N_S$. What is the *minimum* amount of memory (pages in buffer pool) necessary to perform a hash join and achieve best-case performance?

You can assume you have a "perfect" hash function and no data skew, so each partition is (roughly) equal in size.

Hint: Your answer should be a function of N_R or N_S such as $(N_R)^2$ or $\sqrt{N_S}$, $\log(N_R)$, and it can be approximate.

Cost analysis:

$2 \times N_R +$

$2 \times N_S +$

$N_R + N_S$ *(best case: assumes for each pair, $P_i(R)$ and $P_i(S)$, at least one partition fits in memory)*

Sort-merge join

```
Sort R on A: write results to temp1
Sort S on B: write results to temp2
Merge temp1 and temp2
```

Cost analysis:

$2 \times N_R \times$ (no. of passes on R) +
$2 \times N_S \times$ (no. of passes on S) +
$N_R + N_S$ (best case cost of merge)

Cost (on example): *no. of passes is 2 on each relation*
$5 \times (500 + 1000) = 7{,}500$ I/Os $\approx$ **1.25 minutes**

Poll

Instructions: *I will give you 1-2 minutes to think on your own.*
Vote 1.
Then you will discuss w/ neighbor (1 min).
Vote 2.
Then we'll discuss as class.

Cost analysis of sort-merge:
$2 \times N_R \times$ (no. of passes on R) +
$2 \times N_S \times$ (no. of passes on S) +
$N_R + N_S$ (best case cost of merge)

Correct answer: D

Cost analysis of hash join:
$2 \times N_R$ +
$2 \times N_S$ +
$N_R + N_S$ *(best case: assumes for each pair, $P_i(R)$ and $P_i(S)$, at least one partition fits in memory)*

Compare sort-merge and hash join. Which of the following are true? Be sure to be able to explain your answer!

A. Hash join is preferable when one relation is huge and the other is really small (fits in memory).

B. Sort-merge is preferable when the join keys exhibit *skew* (some values appear many, many times).

C. Sort-merge is preferable when both inputs are sorted by join key.

D. All of the above.

Index nested loops

```
// assume index over tuples of S on attribute B
for each t in R:
  probe index for t.A
  for matching tuples t':
    add join(t,t') to result
```

Cost analysis:

$N_R + (p_R \times N_R) \times c$

where *c* is the cost of probing index (see prev. lectures)

Cost (on example): assume that R is outer and c = 2

$1000 + (100 \times 1000) \times 2 = 201{,}000$ I/Os ≈ **30 minutes**