THE UNIVERSITY OF MELBOURNE

INFO20003 Database Systems

Xiuge Chen

Tutorial 7

2021.04.19

Agenda

1. **Notice - 5min**
2. **Effect of index on selection operator - 10min**
3. **Matching index - 10min**
4. **Cost estimation for different joins - 30 min**

THE UNIVERSITY OF MELBOURNE

Notice

1. Assignment 2 has released - LMS Assignments
2. due date: **6:00pm Friday 30 April**
3. Tips:
   - Follow the submission instruction and format
   - Try SQL practice first - LMS Practice on your own / Lab
   - Might involve some SQL functions not taught - Google
   - Complex queries - break down into sub tasks - nest
   - Always check solutions manually

Effect of index on selection operator

1. Consider a relation R (a,b,c,d,e) containing 5,000,000 records, where each data page of the relation holds 10 records. R is organized as a sorted file with secondary indexes. Assume that R.a is a candidate key for R, with values lying in the range 0 to 4,999,999, and that R is stored in R.a order. For each of the following relational algebra queries, state which of the following three approaches is most likely to be the cheapest:

- **Access the sorted file of R directly.**
- **Use a B+ tree index on attribute R.a.**
- **Use a hash index on attribute R.a.**

Effect of index on selection operator

a. $\sigma_{a < 50000}(R)$

- **Access the sorted file of R directly.**
- **Use a B+ tree index on attribute R.a.**
- **Use a hash index on attribute R.a.**

Effect of index on selection operator

a. $\sigma_{a < 50000}$ (R)

- **Access the sorted file of R directly.**
- **Use a B+ tree index on attribute R.a.**
- **Use a hash index on attribute R.a.**

The sorted file over R is preferred in this case –

just **start from the beginning** producing results directly.

Effect of index on selection operator

b. $\sigma_{a = 50000}(R)$

- **Access the sorted file of R directly.**
- **Use a B+ tree index on attribute R.a.**
- **Use a hash index on attribute R.a.**

Effect of index on selection operator

b. $\sigma_{a = 50000}(R)$

- **Access the sorted file of R directly.**
- **Use a B+ tree index on attribute R.a.**
- **Use a hash index on attribute R.a.**

This is an **equality query**.

A hash index will be the most cost-effective option in this case.

Effect of index on selection operator

c. $\sigma_{a>50000 \wedge a<50010}(R)$

- **Access the sorted file of R directly.**
- **Use a B+ tree index on attribute R.a.**
- **Use a hash index on attribute R.a.**

Effect of index on selection operator

c. $\sigma_{a>50000 \wedge a<50010}(R)$

- **Access the sorted file of R directly.**
- **Use a B+ tree index on attribute R.a.**
- **Use a hash index on attribute R.a.**

This is a **range query** which **does not begin** at the start of the sorted file.

A B+ tree index should be the cheapest of all the options

THE UNIVERSITY OF MELBOURNE

Effect of index on selection operator

Any questions?

Matching index

2. Matching index

Consider the following schema for the Sailors relation:

Sailors (sid INT, sname VARCHAR(50), rating INT, age DOUBLE)

For each of the following indexes, list whether the index matches the given selection conditions and briefly explain why.

Matching index

I. A **B+ tree** index on the search key (Sailors.sid)

a. $\sigma_{\text{Sailors.sid} < 50{,}000}$ **(Sailors)**

Match, primary conjuncts are: *Sailors.sid < 50,000*

b. $\sigma_{\text{Sailors.sid} = 50{,}000}$ **(Sailors)**

Match, primary conjuncts are: *Sailors.sid = 50,000*

Matching index

II. A **hash index** on the search key (Sailors.sid)

a. $\sigma_{\text{Sailors.sid} < 50{,}000}$ **(Sailors)**

No match, range queries cannot be applied to a hash index.

b. $\sigma_{\text{Sailors.sid} = 50{,}000}$ **(Sailors)**

Match, primary conjuncts are: *Sailors.sid = 50,000*

THE UNIVERSITY OF MELBOURNE

Matching index

III. A **B+ tree** index on the search key (Sailors.rating, Sailors.age)

a. $\sigma_{\text{Sailors.rating} < 8 \land \text{Sailors.age} = 21}$ **(Sailors)**

Match, primary conjuncts are *Sailors.rating < 8* and *Sailors.rating < 8 ∧ Sailors.age = 21*

b. $\sigma_{\text{Sailors.rating} = 8}$ **(Sailors)**

Match, primary conjuncts are: *Sailors.rating = 8*

c. $\sigma_{\text{Sailors.age} = 21}$ **(Sailors)**

No match. The index on (Sailors.rating, Sailors.age) is primarily sorted on Sailors.rating, so the **entire relation** would need to be searched to find those sailors with a particular Sailors.age value.

THE UNIVERSITY OF MELBOURNE

Matching index

Any questions?

Cost estimation for different joins

Consider the join R $\bowtie_{R.a = S.b}$ S, given the following information about the relations to be joined:

Relation R contains 10,000 tuples and has 10 tuples/page.
Relation S contains 2,000 tuples and also has 10 tuples/page.
Attribute b of relation S is the primary key for S.
Both relations are stored as simple heap files.
Neither relation has any indexes built on it.
52 buffer pages are available.

The cost metric is the number of page I/Os unless otherwise noted and the cost of writing out the result should be uniformly ignored.

Let $M=10{,}000 / 10$ = 1000 be the number of pages in R

$N=2{,}000 / 10$ = 200 bet he number of pages in S, and

B = 52 be the number of buffer pages available.

Cost estimation for different joins

a. What is the cost of joining R and S using the **page-oriented Simple Nested Loops** algorithm? What is the minimum number of buffer pages (in memory) required in order for this cost to remain unchanged?

The basic idea of page-oriented nested loops join is:

1. do a page by page scan of the **outer** relation

2. for each outer page, do a page-by-page scan of the **inner** relation.

The cost can be minimised by selecting the smaller relation as the outer relation:

Cost estimation for different joins

a. What is the cost of joining R and S using the **page-oriented Simple Nested Loops** algorithm? What is the minimum number of buffer pages (in memory) required in order for this cost to remain unchanged?

Total cost = (# of pages in outer) + (# of pages in outer × # of pages in inner)

$$= N + (N \times M)$$

$$= 200 + (200 \times 1000)$$

$$= 200{,}200$$

total 3 buffer pages are required. (one input R, one input S, one output)

Cost estimation for different joins

b. What is the cost of joining R and S using the **Block Nested Loops** algorithm? What is the minimum number of buffer pages required in order for this cost to remain unchanged?

In block nested loops join

1. the outer relation is read in blocks (groups of pages that will fit into whatever buffer pages are available)

2.for each block, do a page-by-page scan of the inner relation.

The outer relation is scanned once, and the inner relation is scanned only once for the outer block.

Cost estimation for different joins

b. What is the cost of joining R and S using the **Block Nested Loops** algorithm? What is the minimum number of buffer pages required in order for this cost to remain unchanged?

# of blocks = ceil(#pages in outer / (B - 2)) = ceil(200 / 50) = 4

Total cost = (# of pages in outer) + (# of blocks × # of pages in inner)

= 200 + (4 × 1000)

= 4200

The minimum number of buffer pages is 52 (FULL) for this cost.

Cost estimation for different joins

c. What is the cost of joining R and S using the **Sort-Merge Join** algorithm? Assume that the external merge sort process can be completed in 2 passes.

external merge sort:

1. The initial sorting pass will split R into 20 runs of 50 buffer pages. After that, these pages will be written to the disk.

2. These sorted pages will be read again to be compared across runs and combine the results. The resulting sorted pages will be written to disk.

3. Similarly, S will split into 4 runs of approximately 50 buffer pages and follow the same process as R.

The general formula for external merge sort is $2N \times$ # of passes.

Cost estimation for different joins

c. What is the cost of joining R and S using the **Sort-Merge Join** algorithm? Assume that the external merge sort process can be completed in 2 passes.

Cost of sorting R = 2 × # of passes × # of pages of R

= 2 × 2 × 1000 = 4000

Cost of sorting S = 2 × 2 × 200 = 800

Cost of merging R and S = # of pages read of R + # of pages read of S

= 1000 + 200 = 1200

Total cost = Cost of sorting R + Cost of sorting S + Cost of merging R and S

= 4000 + 800 + 1200 = 6000

Cost estimation for different joins

d. What is the cost of joining R and S using the **Hash Join** algorithm?

In hash join

1. each relation is partitioned

2. join is performed by "matching" elements from corresponding partitions.

Cost estimation for different joins

d. What is the cost of joining R and S using the **Hash Join** algorithm?

$$\begin{aligned}\text{Total cost} &= 3(M + N) \\ &= 3(1000 + 200) \\ &= 3600\end{aligned}$$

Cost estimation for different joins

e. What would the lowest possible I/O cost be for joining R and S using any join algorithm, and how much buffer space would be needed to achieve this cost? Explain briefly.

Hint: each relation was read only once

1. storing the entire smaller relation in memory

2. reading in the larger relation page by page and for each tuple in the larger relation we search the smaller relation (which exists entirely in memory) for matching tuples.

Block nested loop Join!

Total cost = $M + N = 1200$

minimum number of buffer pages = $\min\{M, N\} + 1 + 1 = 202$.

Cost estimation for different joins

Any questions?