Database Systems

Tutorial Week 7

Objectives

I. Effect of index on selection operator
II. Matching index
III. Cost estimation for different joins

Exercises (5 minutes)

1. **Question about the effect of index on selection:**

   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.

   Queries:

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

Exercises

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

- Sorted file over R
- Start from the beginning of the file and stop when a = 50,000
- If we choose B+ tree index, we first have to find where a = 50,000 and then find records to the left (lower in value)

Exercises

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

- Equality query
- Hash-index will be the most cost-effective

Exercises

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

- Range query that doesn't begin at the start of the sorted file
- B+ tree index will be the cheapest
- With sorted file, we'd have to do 50,000 comparisons before even getting to a = 50,000

Matching Predicates

- Queries usually have multiple conditions called **predicates**
- What's the predicate for Q2a?
    - Sailors.sid < 50,000
- A B-tree index matches predicate(s) that involve attributes in a **prefix of the search key**
- Say you have an index on <a, b, c>
- This index will match conditions/predicates on (a, b, c), (a, b) and (a), but not (b), (b, c) or (c)
- Any combination of predicates that involve attributes in a **prefix** are called **matching predicates**
- An index can be used to speed up an analysis over matching predicates only

Exercises (10 minutes)

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.

- A B+ tree index on the search key (Sailors.sid)
  a. $\sigma_{\text{Sailors.sid} < 50,000}$ (Sailors)
  b. $\sigma_{\text{Sailors.sid} = 50,000}$ (Sailors)
- A hash index on the search key (Sailors.sid)
  c. $\sigma_{\text{Sailors.sid} < 50,000}$ (Sailors)
  d. $\sigma_{\text{Sailors.sid} = 50,000}$ (Sailors)
- A B+ tree index on the search key (Sailors.rating, Sailors.age)
  e. $\sigma_{\text{Sailors.rating} < 8 \wedge \text{Sailors.age} = 21}$ (Sailors)
  f. $\sigma_{\text{Sailors.rating} = 8}$ (Sailors)
  g. $\sigma_{\text{Sailors.age} = 21}$ (Sailors)

Exercises

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

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

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

With B+ tree indexes, we can do range checks and equality checks

a. Yep! Matching predicates are Sailors.sid < 50,000

b. Yep! Matching predicates are Sailors.sid = 50,000

Exercises

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

c. $\sigma_{Sailors.sid < 50,000}(Sailors)$

d. $\sigma_{Sailors.sid = 50,000}(Sailors)$

With hash indexes, we can do equality checks

c. Nope! Range queries can’t be applied to a hash index :(

d. Yep! Matching predicates are Sailors.sid = 50,000

Exercises

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

e. $\sigma_{\text{Sailors.rating} < 8 \wedge \text{Sailors.age} = 21}(\text{Sailors})$

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

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

e. Yep! Matching predicates are Sailors.rating < 8, and
Sailors.rating < 8 ∧ Sailors.age = 21

f. Yep! Matching predicates are Sailors.rating = 8

g. Nope! The index on (Sailors.rating, Sailors.age) is primarily sorted on Sailors.rating, so you'd need to search the entire relation to find sailors with a particular Sailors.age value

Exercises (15 mins)

3. Question about the cost analysis of 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.

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?

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?

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.

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

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.

Exercises

Step 1: always find how many pages you need for each relation

Number of pages = number of tuples / number of tuples/page

Let M be the number of pages in R

- M = 10,000 / 10 = 1,000

Let N be the number of pages in S

- N = 2,000 / 10 = 200

Let B be the number of buffer pages available

- B = 52

Exercises

a. What is the cost of joining R and S using the **page-oriented Simple Nested Loops** algorithm?

- Need to know page-oriented Simple Nest Loop in detail
- Do a page-by-page scan of the outer relation
- For each outer page, do a page-by-page scan of the inner relation
- How do we minimise the cost of joining R and S?
    - Formula is: NPages(outer) + NPages (outer) × NPages(inner)
    - Select the smaller relation as the outer relation
    - S has 200 pages, R has 1,000 pages, so in this case, choose S as the outer relation

Exercises

a. What is the cost of joining R and S using the **page-oriented Simple Nested Loops** algorithm?

- Cost = NPages(outer) + NPages(outer) × NPages(inner)
- S = outer, R = inner
- Cost
  = 200 + (200 × 1,000)
  = 200,200 I/O

Exercises

a. What is the minimum number of buffer pages (in memory) required in order for this cost to remain unchanged?

- In the page-oriented Simple Nested Loops algorithm, we don’t use multiple buffers at a time
- Minimum number of buffer pages is:
  - One buffer page for the left input
  - One buffer page for the right input
  - One buffer page to store the output/result
  - Total = 3 buffer pages required

Exercises

b. What is the cost of joining R and S using the **Block Nested Loops** algorithm?

- Need to know Block Nested Loop in detail
- Outer relation is read in "blocks"
  - "Blocks" are groups of pages that will fit into whatever buffer pages are available
- For each block, do a page-by-page scan of the inner relation
- Each page of the outer relation is scanned once
- Each page of the inner relation is scanned once per block
- Cost = NPages(outer) + NBlocks × NPages(inner)
- Use one buffer page for scanning the inner relation
- Use one buffer page to store the output
- Use all other buffer pages to hold the blocks of the outer relation
- NBlocks = ceil(NPages(outer) / B–2)
  - Ceiling means you round up to the nearest integer

Exercises

b. What is the cost of joining R and S using the **Block Nested Loops** algorithm?

- Cost
  = NPages(outer) + ceil(NPages(outer) / B–2) × NPages(inner)
  = 200 + ceil(200/50) × 1,000
  = 200 + (4 × 1,000)
  = 4,200 I/O

Exercises

b. What is the minimum number of buffer pages required in order for this cost to remain unchanged?

- Cost = NPages(outer) + NBlocks × NPages(inner)
- NBlocks = ceil(NPages(outer) / B–2)
- Fewer buffer pages → denominator decreases → NBlocks increases → overall cost increases
- So we can’t use fewer buffer pages
- Have to use all 52 i.e. minimum = 52

Exercises

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**.

- Don't need to know Sort-Merge Join algorithm in too much detail
- Cost

  = NPages(outer) + NPages(inner) — Cost of merging outer and inner relations

  + 2 × NPages(outer) × num_passes(outer) — Cost of accessing and sorting outer relation

  + 2 × NPages(inner) × num_passes(inner) — Cost of accessing and sorting inner relation

  = 200 + 1000 + (2 × 200 × 2) + (2 × 1,000 × 2)

  = 6,000 I/O

Exercises

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

- Don't need to know Hash Join algorithm in too much detail
- In hash join, each relation is partitioned and then the join is performed by "matching" elements from corresponding partitions
- Cost
  $= 3 \times (\text{NPages(outer)} + \text{NPages(inner)})$
  $= 3 \times (200 + 1{,}000)$
  $= 3{,}600$ I/O

Exercises

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?

- Optimal cost achieved if each relation is read only once
- We could store entire smaller relation in memory
- We could then read in the larger relation page by page
  - For each tuple in the larger relation, we could search the smaller relation (which exists entirely in memory) for matching tuples
- Total cost = NPages(smaller relation) + NPages(bigger relation) = 200 + 1,000 = 1,200 I/O
- Buffer would have to:
  - Hold the smaller relation
  - Have one buffer page to read in the larger relation
  - Have one buffer page to store the output
- Minimum number of buffer pages is NPages(smaller relation) + 1 + 1 = 200 + 1 + 1 = 202

Week 7 Lab

- Canvas → Modules → Week 7 → Lab → L07 SQL 3 (PDF)
- Objectives:
    - Practice further joins involving three and four tables
    - Understand CASE statements and the UNION clause
    - Develop complex SQL queries using derived tables and views
        - Note you're ***not*** allowed to use views for assignment 2
    - Create and understand relational divides using EXISTS and NOT EXISTS
        - Video on Canvas
- Breakout rooms, "ask for help" button if you need help or have any questions