Query Processing

Introduction to Databases
CompSci 316 Spring 2020
Announcements (Thu., Mar 05)

• **Next week: Spring break, no class!**
  • No project update needed
  • No office hours (email the instructor if you would like to talk)

• **Lab-2 today at the end for about 20 mins**
  • Can submit by tomorrow (Fri) night, but submit early!
  • 10% extra credit for submitting all questions correctly before class ends.
  • Can discuss but everyone submits their own answers

• Check out my email on sakai about project coordination and updates
Clustered vs. Unclustered Index

- SELECT * FROM USER WHERE age = 50
  - Assume 12 users with age = 50
  - Assume one data page can hold 4 User tuples

- What happens if the index is unclustered?
  - Cost to access data pages can be $12$
- What happens if the index is clustered?
  - Cost to access data pages can be $3$ or $4$. 

- Why?
Hash vs. Tree Index

- Hash indexes can only handle equality queries
  - SELECT * FROM R WHERE age = 5 (requires hash index on (age))
  - SELECT * FROM R, S WHERE R.A = S.A (requires hash index on R.A or S.A)
  - SELECT * FROM R WHERE age = 5 and name = ‘Bart’ (requires hash index on (age, name))

- Cannot handle range queries or prefixes
  - SELECT * FROM R WHERE age >= 5
  - need to use tree indexes (more common)
  - Tree index on (age), or (age, name) works, but not (name, age) – why?

- But are more amenable to parallel processing
  - later hash-based join

- Performance depends on how good the hash function is (whether the hash function distributes data uniformly and whether data has skew)

- Details of hash-based dynamic index (extendible hashing, linear hashing) not covered in this class
Trade-offs for Indexes

• Should we use as many indexes as possible?
Trade-offs for Indexes

• Should we use as many indexes as possible?

• Indexes can make
  • queries go faster
  • updates slower

• Require disk space, too
Query Processing Overview

• Many different ways of processing the same query
  • Scan? Sort? Hash? Use an index?
  • All have different performance characteristics and/or make different assumptions about data
• Best choice depends on the situation
  • Implement all alternatives
  • Let the query optimizer choose at run-time
Notation

• Relations: $R, S$
• Tuples: $r, s$
• Number of tuples: $|R|, |S|$
• Number of disk blocks: $B(R), B(S)$
• Number of memory blocks available: $M$
• Cost metric
  • Number of I/O’s
  • Memory requirement

Recall our disk-memory diagram
On board!
• How do we implement selection and projection?

• Ideas? (discuss with neighbors)

• Cost?
  • (page I/O -- in terms of B(R), |R| etc.)

• Memory requirement?
Scanning-based algorithms
Table scan

• Scan table $R$ and process the query
  • Selection over $R$
  • Projection of $R$ without duplicate elimination

• I/O’s: $B(R)$
  • Trick for selection: stop early if it is a lookup by key

• Memory requirement: 2

• Not counting the cost of writing the result out
  • Same for any algorithm!
  • Maybe not needed—results may be pipelined into another operator
• How do we implement $\text{Join}$?

• Ideas? (discuss with neighbors)

• Cost?
  • (page I/O -- in terms of $B(R)$, $|R|$ etc.)

• Memory requirement?
Nested-loop join

\[ R \bowtie_p S \]

- For each block of \( R \), and for each \( r \) in the block:
  - For each block of \( S \), and for each \( s \) in the block:
    - Output \( rs \) if \( p \) evaluates to true over \( r \) and \( s \)
- \( R \) is called the outer table; \( S \) is called the inner table
- I/O’s: \( B(R) + |R| \cdot B(S) \)
- Memory requirement: 3

Improvement: block-based nested-loop join
Block-based Nested Loop Join

• $R \bowtie_p S$

• R outer, S inner

• For each block of $R$, for each block of $S$:
  For each $r$ in the $R$ block, for each $s$ in the $S$ block: ...
  • I/O’s: $B(R) + B(R) \cdot B(S)$
  • Memory requirement: same as before