

Relational Operator Evaluation

EECS 484

Winter 09

Basics

- Cost = page I/Os with disk
- Writing back result common, so may be ignored

SELECTION

```
SELECT *  
FROM Table T  
WHERE T.attr = 'val'
```

1. No index, unsorted?
2. No index, sorted?
3. B+ tree index? – clustered?
4. Hash index?

When to use a B+tree index

- Consider
 - A relation with 1M tuples
 - 100 tuples on a page
 - 500 (key, rid) pairs on a page

data pages
= $1M/100 = 10K$ pages

leaf idx pgs
= $1M / (500 * 0.67)$
~ 3K pages

	1% Selection	10% Selection
Clustered	30 + 100	300 + 1000
Non-Clustered	30 + 10,000	300 + 100,000
NC + Sort Rids	30 + (~ 10,000)	300 + (~ 10,000)

SELECTION

- No disjunction
 - File scan
 - Single index
 - Multiple index? – intersect rids
- Disjunctions present
 - File scan required for any clause?
 - Multiple index? – union rids

PROJECTION

- Duplicates allowed?
- Sort-based
 - Get tuples with required attributes
 - Sort
 - Duplicate elimination [adjacent]
 - Modification
 - Purge in Pass 0
 - Eliminate duplicates when merging

PROJECTION

- Hash-based
 - Hash relation into B-1 partitions
 - Read in partition
 - For each, re-hash, eliminate duplicates, write back
- Indexes
 - If all attributes present, 'index only'

JOIN

- Nested Loops join

foreach tuple r in R do

 foreach tuple s in S do

 if r.sid == s.sid then add <r, s> to result

$$\text{Cost} = |R| + ||R|| * |S|$$

Should R be the Smaller/Larger relation?

- Page Nested Loops join

$$|R| + |R| * |S|$$

JOIN

- Block Nested Loops join

$$|R| |S| * \left\lceil \frac{|R|}{B-2} \right\rceil$$

- Index Nested Loops join

$$|R| + (||R|| * \text{cost of finding matching S tuples})$$

JOIN

- Sort-Merge Join
 - **Sort R and S**
 - **Increment pointers till matching tuples found**
 - **Take care of duplicates**
- Cost
 - $|R| \log |R| + |S| \log |S| + (|R| + |S|)$
 - Refinement: $3(|R| + |S|)$

JOIN

- Hash Join
 - Build hashes on R, S
 - Read in smaller partition and rehash
 - Probe other relation
- Cost
 - $3(|R|+|S|)$
- Sort vs Hash
 - Sort skew resistant, result sorted
 - If B between $\sqrt{|R|}$ and $\sqrt{|S|}$, hash economical

SET OPERATIONS

- Sort
- Hash

Consider the join $R \text{ JOIN } (R.a=S.b) S$.

Ignore the cost of writing out the result.

Relation R contains 10,000 tuples and has 10 tuples per page.

Relation S contains 2000 tuples and also has 10 tuples per 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.

1. What is the cost of joining R and S using a page-oriented simple nested loops join? What is the minimum number of buffer pages required for this cost to remain unchanged?
2. What is the cost of joining R and S using a block nested loops join? What is the minimum number of buffer pages required for this cost to remain unchanged?
3. What is the cost of joining R and S using a sort-merge join?
4. What is the cost of joining R and S using a hash join?
5. What would be the lowest possible I/O cost for joining R and S using *any join* algorithm, and how much buffer space would be needed to achieve this cost?
6. How many tuples does the join of R and S produce, at most?

Table 'Executives' has attributes *ename*, *title*, *dname*, and *address*; all are *string* fields of the same length.

The *ename* attribute is a candidate key.

The relation contains 10,000 pages.

There are 10 buffer pages.

Index record is $1/4^{\text{th}}$ a data record for single key and $1/2$ a data record for 2-key indexes

Consider the following query:

```
SELECT E.title, E.ename FROM Executives E WHERE E.title='CFO'
```

Assume that only 10% of Executives tuples meet the selection condition.

What is the cost of the best plan, if you have

- (a) a clustered B+ tree index on *title*.
- (b) an unclustered B+ tree index on *title*.
- (c) a clustered B+ tree index on *ename*.
- (d) a clustered B+ tree index on *address*.
- (e) a clustered B+ tree index on *ename*, *title*.