Outline

- Sorting
- Evaluation of joins
- Evaluation of other operations

Some Common Techniques

- Algorithms for evaluating relational operators use some simple ideas extensively:
  - Indexing: Can use WHERE conditions to retrieve small set of tuples (selections, joins)
  - Iteration: Sometimes, faster to scan all tuples even if there is an index. (And sometimes, we can scan the data entries in an index instead of the table itself.)
  - Partitioning: By using sorting or hashing, we can partition the input tuples and replace an expensive operation by similar operations on smaller inputs.

* *Watch for these techniques as we discuss query evaluation!*

Schema for Examples

Sailors (*sid*: integer, *sname*: string, *rating*: integer, *age*: real)
Reserves (*sid*: integer, *bid*: integer, *day*: date, *rname*: string)

- Reserves:
  - Each tuple is 40 bytes long,
  - 100 tuples per page,
  - 1000 pages.
- Sailors:
  - Each tuple is 50 bytes long,
  - 80 tuples per page,
  - 500 pages.

Equality Joins With One Join Column

```
SELECT  *
FROM    Reserves R1, Sailors S1
WHERE   R1.sid=S1.sid
```

- In algebra: R $\bowtie$ S. Common relational operation!
  - R X S is large; R X S followed by a selection is inefficient.
  - Must be carefully optimized.
- Assume: M pages in R, p_R tuples per page, N pages in S, p_S tuples per page.
  - In our examples, R is Reserves and S is Sailors.
- We will consider more complex join conditions later.
- *Cost metric*: # of I/Os. We will ignore output costs.

Simple Nested Loops Join

foreach tuple r in R do
    foreach tuple s in S do
        if $r_i == s_j$ then add <r, s> to result

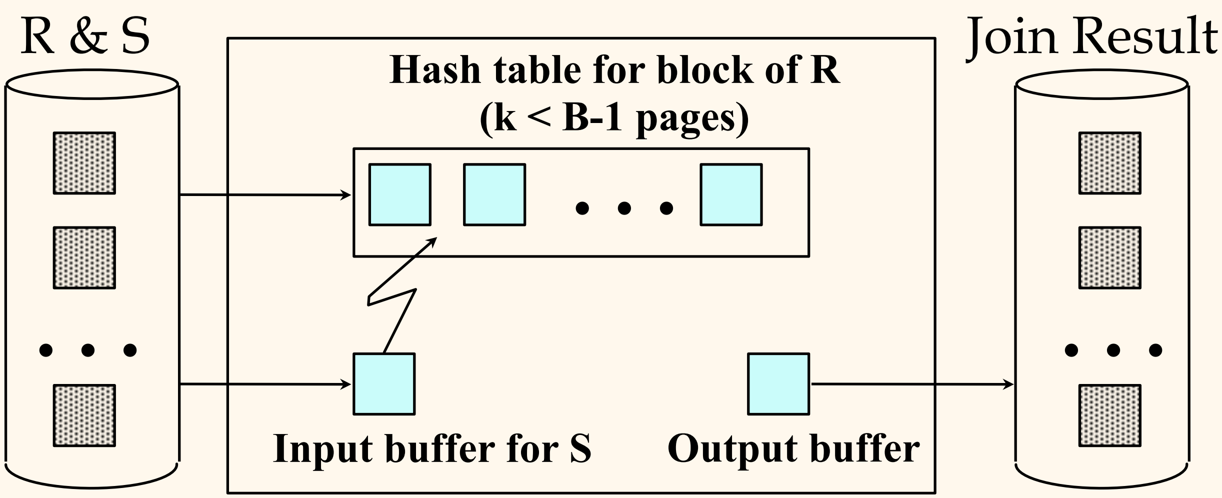
- For each tuple in the *outer* relation R, we scan the entire *inner* relation S.
  - Cost: $M + p_R * M * N = 1000 + 100*1000*500 = 1{,}000 + (5 * 10^7)$ I/Os.
  - Assuming each I/O takes 10 ms, the join will take about 140 hours!

Page-Oriented Nested Loops Join

- For each *page* of R, get each *page* of S, and write out matching pairs of tuples <r, s>, where r is in R-page and S is in S-page.
  - Cost: M + M * N = 1000 + 1000*500 = 501,000 I/Os.
  - Assuming each I/O takes 10 ms, the join will take about 1.4 hours.
- Choice of the *smaller* relation as the *outer*
  - If smaller relation (S) is outer, cost = 500 + 500*1000 = 500,500 I/Os.

Block Nested Loops Join

- Take the smaller relation, say R, as outer, the other as inner.
- Use one buffer for scanning the inner S, one buffer for output, and use all remaining buffers to hold ``block'' of outer R.
  - For each matching tuple r in R-block, s in S-page, add <r, s> to result.
  - Then read next page in S, until S is finished.
  - Then read next R-block, scan S…



Examples of Block Nested Loops

- Cost: Scan of outer + #outer blocks * scan of inner
  - #outer blocks = $\lceil$ # pages of outer / block size $\rceil$
  - Given available buffer size B, block size is at most B-2.
  - M + N * $\lceil$ M / B-2 $\rceil$
- With Sailors (S) as outer, a block has 100 pages of S:
  - Cost of scanning S is 500 I/Os; a total of 5 *block*s.
  - Per block of S, we scan Reserves; 5*1000 I/Os.
  - Total = 500 + 5 * 1000 = 5,500 I/Os.

Index Nested Loops Join

foreach tuple r in R do
    foreach tuple s in S where $r_i == s_j$ do
        add <r, s> to result

- If there is an index on the join column of one relation (say S), can make it the <u>inner</u> and exploit the index.
  - Cost: M + ($(M*p_R)$ * cost of finding matching S tuples)
- For each R tuple, cost of probing S index is about 1.2 for hash index, 2-4 for B+ tree. Cost of then finding S tuples (assuming Alt. (2) or (3) for data entries) depends on clustering.
  - Clustered index: 1 I/O (typical).
  - Unclustered: upto 1 I/O per matching S tuple.

Examples of Index Nested Loops

- Hash-index (Alt. 2) on *sid* of Sailors (as inner):
  - Scan Reserves: 1000 page I/Os, 100*1000 tuples.
  - For each Reserves tuple: 1.2 I/Os to get data entry in index, plus 1 I/O to get (the exactly one) matching Sailors tuple.
  - Total: 1000+ 100*1000*2.2 = 221,000 I/Os.
- Hash-index (Alt. 2) on *sid* of Reserves (as inner):
  - Scan Sailors: 500 page I/Os, 80*500 tuples.
  - For each Sailors tuple: 1.2 I/Os to find index page with data entries, plus cost of retrieving matching Reserves tuples. If uniform distribution, 2.5 reservations per sailor (100,000 / 40,000). Cost of retrieving them is 1 or 2.5 I/Os (cluster?).
  - Total: 500+80*500*(2.2~3.7) = 88,500~148,500 I/Os.

Sort-Merge Join $(R \bowtie_{i=j} S)$

- (1) Sort R and S on the join column, (2) Merge them (on join col.), and output result tuples.
- Merge: repeat until either R or S is finished
  - *Scanning*: Advance scan of R until current R-tuple>=current S tuple, advance scan of S until current S-tuple>=current R tuple; do this until current R tuple = current S tuple.
  - *Matching*: Now all R tuples with same value in Ri (*current R group*) and all S tuples with same value in Sj (*current S group*) match; output <r, s> for all pairs of such tuples.
- R is scanned once; each S group is scanned once per matching R tuple. (Multiple scans of an S group are likely to find needed pages in buffer.)

Example of Sort-Merge Join

sid	sname	rating	age
22	dustin	7	45.0
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

sid	bid	day	rname
28	103	12/4/96	guppy
28	103	11/3/96	yuppy
31	101	10/10/96	dustin
31	102	10/12/96	lubber
31	101	10/11/96	lubber
58	103	11/12/96	dustin

- Cost: M log M + N log N + (M+N)
  - The cost of merging, M+N, could be M*N (very unlikely!)
  - M+N is guaranteed in *foreign key join* (why?)
  - As with sorting, log M and log N are small numbers, e.g., 3, 4.
- With 35, 100 or 300 buffer pages, both Reserves and Sailors can be sorted in 2 passes; total join cost: 7500.

  (*BNL cost: 2500 (B=300), 5500 (B=100), 15000 (B=35)*)