Simple Sort based Join

One potential problem:

What if for some joining values the number of tuples is very large.

Worst case: what if there is only one value of att to join in both relations.

Example:

Assume $R(a, b)$ $S(a, c)$

$R \bowtie S$.

Assume S only as $a = 1$, and $|S| > M$.

Assume R has n tuples with $a = 1$.

$\Rightarrow$ We need to join each tuple of R with $a = 1$ with <u>every</u> tuple of S.

Hence we need to read S n times!!

In general, assume that the maximum number of tuples with same joining attribute requires less than M blocks of memory.

Assume $R(X,Y) \bowtie S(Y,Z)$ Y is set of att.

1) Sort R using 2PMMS using Y as sort key. Write result to disk.
2) Sort S the same way.
3) Merge R and S

Repeat until done

$y \leftarrow$ min value of att set Y in both R and S.

if y is only in one relation remove all tuples with y.

else

Do join of tuples $\sigma_{Y=y}$ R and $\sigma_{Y=y}$ S using M buffers

(See previous algorithms).

In general, Step 3 one pass unless there is one value x of Y

s.t $(\sigma_{Y=x} R \cup \sigma_{Y=x} S) > M$.

Total cost:

Step 1: $3B(R) + B(R)$

↑ sort ↑ write back

Step 2: $3B(S) + B(S)$

Step 3: $B(S) + B(S)$.

Total:

$5(B(R) + B(S))$

as long as:

$B(R) \leq M^2 \quad B(S) \leq M^2$

and

for any common joining attributes we can fit all tuples in M.

What is the worst case (unusual)?

Use block-based loop join.

$$\text{Cost} = B(S) + \frac{B(R) \cdot B(S)}{M}$$

If we do not have to worry about a large number of tuples with the same joining attributes we can do the join in the second pass of the sort:

Alg:

1) As in the first pass of 2PMMS sort sections of size M of both R and S.
2) Repeat until done:
   reading one block of each section.
   (# sections < M)
   join tuples with the smallest joining attributes.
   (We might need few extra blocks for some values but it is likely that there are some unused blocks).

Total cost: $3(B(R)+B(S))$.

Memory Required: $B(R)+B(S) \leq M^2$

Sort based algorithms:

Operator	M. Required	Cost
τ, γ, δ	$B < M^2$	$3B$
$\cup, \cap, -$	$B(R)+B(S) < M^2$	$3(B(R)+B(S))$
$\bowtie$	$\max(B(R), B(S)) < M^2$	$5(B(R)+B(S))$
$\bowtie$	$B(R)+B(S) < M^2$	$3(B(R)+B(S))$

Index Based Selection.

See indexing.

Bitmap index scan.

When an index is unclustered the cost of index is too large:

$$h-1+\left\lceil \frac{\#\text{matching tuples}}{\#\text{reads in index block}} \right\rceil + \#\text{matching tuples}.$$

We might read a block many times.

Instead:

- Allocate a bitmap of size B(R)
- Scan index as in usual index-based selection but instead of reading block, mark bit in bitmap for corresponding blocked in heap.
- Read every block marked in bitmap output tuples that satisfy condition.

Cost:

$$h-1+\left\lceil \frac{\#\text{matching tuples}}{\#\text{records in index block}} \right\rceil + \text{blocks expected to contain matching tuples}$$

Worst case:

$$h-1+\left\lceil \frac{\#\text{matching tuples}}{\#\text{records in index block}} \right\rceil + B(R)$$

Requires:

$$\frac{B(R)}{8} < M$$

Bitmap Scan is very useful for range queries that match a large number of tuples.

Ex: $\sigma_{a>5} R$.

But at some point cheaper to do sequential scan on relation (needs only 1 block).

Index Based Join.

Assume $R(X,Y) \bowtie S(Y,Z)$

Y is a set of attributes.

Assume S has index on $S(Y)$

For each tuple in R:

find matching tuples in S using index.

On the average

for each tuple in R

there are

$$\frac{|S|}{V(S,Y)}$$ tuples in S.

Dense index on S:

$$B(R)+|R| \cdot \left[h-1+\left\lceil \frac{\frac{|S|}{V(S,Y)}}{\#\text{records per index block}} \right\rceil + \frac{|S|}{V(S,Y)} \right]$$

Because we usually join few tuples at a time bitmap scan is not very useful.

Sparse Index on S:

$$B(R) + |R| \cdot \left[h + \left\lceil \frac{|S|}{V(S,Y)} \right\rceil \right]$$

(#records per heap block)

What if R is sorted?

We only need to join

$$\frac{|R|}{V(R,Y)} \text{ tuples}$$

We can save $V(R,Y)$ times the cost of reading the index and tuples of S.

it might be worth to sort R before join!!

R might be already sorted
- clustered index on Y
- previous op. in eval plan might sort it.

What if both R and S have an unclustered index?
- Calculate both costs
- Use cheapest