Joins and Sorts

Join

Quite a few approaches/algorithms can be used

Nested Loop Join

To perform the join r $\bowtie$ s:

```
for each tuple t_r in r do
   for each tuple t_s in s do
      if t_r and t_s satisfy join condition
         add (t_r,t_s) to result
   end
end
```

Performance ..

- expensive approach
- every pair of tuples is checked to see if they satisfy join condition
- If one of the relations fits in memory, it is beneficial to use this in the inner loop. (known as the inner relation).

Block Nested Loop Join

Variation on the nested loop-join. Increases efficiency by reducing number of block accesses.

```
for each block B_r in r do
   for each block B_s in s do
      for each tuple t_r in B_r do
           for each tuple t_s in B_s do
              if t_r and t_s satisfy join condition
                  add (t_r,t_s) to result
           end
       end
   end
end
```

Indexed Nested Loop Join

If in a nested loop join, there is an index available for the inner table, replace file scans with index accesses

Merge Join

- If both relations are sorted on the joining attribute, then merge relations.
- Technique is identical to merging two sorted lists (like the merge step in a merge-sort algorithm)
- Much more efficient that a nested join
- Can also be computed for relations not ordered on a joining attribute, but have indexes on joining attribute
- Efficiency?

Hash Join

- Create a hashing function which maps the join attribute(s) to partitions in a range $1 \ldots N$
- For all tuples in r, hash the tuples to H_{ri}
- For all tuples in s, hash the tuples to H_{si}
- For $i = 1$ to N, join partitions $H_{ri} = H_{si}$

Sorting

Sorting

Important operation because:

- if a query specifies ORDER BY
- used prior to relational operators (e.g. Join) to allow more efficient processing of operation

Can sort a relation:

- physically: - actual order of tuples re-arranged on disk
- logically: - build an index and sort index entries

Two main cases:

- where relation to be sorted fits in memory- can then use standard sorting techniques (e.g. quicksort)
- where relation doesn't fit in memory. The most common approach is to use external sort-merge

External Sort Merge - Step 1

```
i := 0;
repeat
   read M blocks of the relation
   sort M blocks in memory
   write sorted data to file Ri
until end of relation

M = number of page frames in main memory buffer
```

External Sort Merge - Step 2

Wish to merge the files from each run in step 1

```
read first block of each Ri into memory

repeat
choose first (in sort order) from pages
write tuple to output
remove tuple from buffer
if any buffer Ri if empty and not eof(Ri)
    read next block from Ri into memory
until all pages empty
```

effectively a N-way merge (extension of idea in the merge step of the merge sort algorithm)