Query Processing I

Tom Kelliher, CS 318

Apr. 22, 2002

1 Administrivia

Announcements

Homework due Wednesday, start of class.

Assignment

Read 13.4–7.

From Last Time

Indexing schemes.

Outline

1. Introduction.
2. Sorting.
3. Computing projections.
4. `SELECT` processing.

Coming Up

Completion of query processing.

2 Introduction

Given a table and one or more access paths, how do we structure queries to optimize performance?

3 Sorting

1. Have to rely on external sorting techniques.
2. Most important objective is to minimize disk I/O.
3. In addition to the `SORT BY` clause, sorting will be necessary when `DISTINCT` is in use. Why?
4. Assumptions: file consists of F pages and we have M memory buffers at our disposal.

External sorting algorithm:

```
// Partial sort on runs of M pages.

do {
   read next M pages from the file;
   sort rows using an in-memory technique;
   write the sorted M pages to a file of their own;
}  until end of file;

// Merge the partial sorts into a final sorted file.
// Open files for reading to maintain a balanced tree organization
//  of runs.

while there is more than one input run {
   open M - 1 runs for reading
   while there is a non-empty run {
      choose the smallest tuple, with respect to sort key, from each
       run;
      output the smallest such tuple and delete it from its run;
   }

   write and close the output file;

   }
}
```

$M-1$ because we need one output buffer.

Rough example: file contains 13 pages, we have three buffers.

Questions:

1. How many page accesses to obtain the partial sorts?
2. How many page accesses to complete a merge step?
3. How many merge steps?
4. How many total page accesses to sort?

4 Computing Projections

1. Eliminating columns might result in duplicates.

   `DISTINCT` clause.

2. Eliminate duplicates by sorting or by hashing.
3. Modifications to basic sort needed to support projections:

   (a) Eliminate unnecessary columns from table when first reading pages during the partial sort step.

   (b) Eliminate duplicate rows while writing new files.

   How will this running time compare to unadorned sort?

4. Sketch of hashing:

   (a) Need one input buffer, so we have $M - 1$ buffers available for hashing.

   (b) Find a hash function which hashes the projection columns onto the buffers.

   (c) Read file. For each row, gather projected columns and hash. Flush each hash buffer to a file when full.

   (d) For each hash file, read, sort, eliminate duplicates, write to final output file.

       Assuming a uniform distribution, each hash file will fit in the M buffers. Quite a large file can be handled this way.

   (e) Total number of page accesses?

5. Union and set difference are similar.

5 SELECT Processing

Preliminaries:

1. Our SELECT model:

   ```
   SELECT *
   FROM R
   WHERE Col1 op1 Val1 AND Col2 op2 VAL2 ... Coln opn Valn;
   ```

2. Access path options when there is only one condition:

   (a) No index — scan the entire file.

       Or, if file is sorted on `Col1`, perform a binary search to get to the first "qualifying" tuple and then scan forward.

Cost?

(b) B^+ tree index on `Col1` — Find first qualifying tuple and then use sibling pointers to scan forward.

Cost? Suppose the file is unclustered? Sort the rids.

(c) Hash index on `Col1` — Find bucket containing `Val1` and scan.

Cost?

3. Access path options when there are n conditions:

(a) No index — scan the entire file.

Or, if file is sorted on several of the `Coli`, perform a binary search to get to the first "qualifying" tuple and then scan forward.

Cost?

(b) B^+ tree index on several columns.

For this to be useful, a subset of the `Coli` must be a prefix of the index columns. Example.

Cost?

(c) Hash index on several columns. All `opi` must be equality. The columns indexed on must be a subset of the `Coli`. Why?

Hash to find buckets, then check buckets for matches.

Cost?