Chapter 10 — The Basics of Query Processing

We did not have time to go over *all* of the materials in the chapter. We talk about the following:

- Sorting
- Projection
- Union
- Intersection
- Selection

As usual, we concentrate on understanding how each operation is working and what can a database programer do to get the best out of the DBMS. We operate under the following assumptions:

- The time needed for the computation done in the memory (of the computer) is often small comparing to the time needed to get the data from external memory into the memory for processing. As such, the cost of each operation is approximated by the number of pages that need to be read and/or write between external memory and main memory.
- The above item is often correct since the amount of data that we have to deal with in a database application is often huge (much more than the capacity of the main memory).

Sorting. Assume that the data file that needs to be sorted occupies F pages and the main memory has M pages. Again, we can imagine the case that F is much larger than M. We discuss the sorting process that is commonly implemented by DBMS. It is divided into two phases:

- *External sorting*: Divide the file into groups of M pages, read each page into the main memory, sort them, and write them back to the external memory. After this step, we have F/M groups of M pages and data in each group is sorted.
- $M-1$-*way merging*: In this phase, we merge the sorted groups into larger groups which are sorted. We do the following (the detail of merging is in the book):
  - 1st step: Merge $M-1$ groups, each group consists of M pages (which are sorted during the external sorting phase); this creates a group of sorted $(M-1)M$ pages; this process continues until we read the last of the F/M groups created in the external sorting phase. After this step, we have $F/[(M-1)M]$ groups, each contains $M(M-1)$ pages of sorted data.
  - 2nd step: Merge $M-1$ groups of the first step, creates group of $(M-1)^2M$ pages of sorted data. We then have $F/[(M-1)^2M]$, each contains $M(M-1)$ pages of sorted data.
  - repeat the above steps until we have only 1 group

At each step, we will need to

- read F pages
- write F pages

Furthermore, the number of steps that we need to do is

$$log_{M-1}(F/M) \cong log_{M-1}F - 1$$

Together with the external sorting, the total cost for sorting is

$$2F \times (log_{M-1}F - 1) + 2F = 2Flog_{M-1}F$$

Projection. Assume that we have a file of F pages and we would like to get some attributes of the file. This question can be answered by the SQL statement

```
Select A1,...,Ak
From   R
```

or

```
Select DISTINCT A1,...,Ak
From   R
```

In either case, we will have to scan through the file (i.e., read F pages), remove the unwanted attributes, and output the data (i.e., write out the result, most likely less than F pages). If we do not remove anything then the cost will be $2F$.

If we need to remove duplicates, the most often used method is to sort the file (according to the attributes `A1,...Ak`) and then remove the duplicate rows. This will cost $2F * log_{M-1}F$ (as the cost of sorting).

Note: Notice that given a table, projection does (quite often) result into a table with duplications. An example for this is the transcript table with the attributes `StudID, CrsCode, Semester, Grade`; projection onto the attributes `CrsCode, Semester` will result in the table with a lot of duplications since it is often the case that there are more than one student in a class.

Union/Intersection. The union or intersection of two files of the size F and G pages, respectively, can be computed by

- sort each file
- merge (during this step, duplicates can be eliminated)

The cost of this operation is: $2F * log_{M-1}F + 2G * log_{M-1}G + 2F + 2G$. One can do some optimization and gets rid of $2F + 2G$ in the above computation though.

Selection. Assume that we need to compute the answer for the question

$$\sigma_{condition}(R)$$

or in SQL:

```
select *
from   R
where  condition
```

Note that $condition$ is a Boolean expression built over the attributes of R and constants. For example, if R is the $Transcript$ relation as above, then

$$CrsCode =' CS482' \ and \ Semester =' F2006' \tag{0.1}$$

or

$$StudID =' 123456789' \tag{0.2}$$

or

$$Semester \geq' F2005' \ and \ Semester \leq' S2006' \tag{0.3}$$

are possible conditions.

The cost of a selection depends on the following information:

- The type of condition in the selection condition (e.g., equality search (as in (0.1) or (0.2) or range search (as in (0.3));
- The type of indices available for use

This question is addressed in detail by the notion of an "*access path*." I ask you to read about it in the book. In general, we can make the decision based on the following information:

- Any condition can be evaluated using a *scan* of the file.
- An equality search can be effectively dealt with by a hash index.
- A range search can be effectively dealt with by a B^+-tree, provided that the attributes involves in the condition is a prefix of the list of attributes of the index.

As an example, the query

```
select T.StudID
from   Transcript T
where  CrsCode = 'CS482' and Semester = 'F2006'
```

can be dealt with effectively using an B^+-tree index on the list of attributes `CrsCode, Semester` or `CrsCode, Semester, StudID`. However, we will need to scan the whole file if the index is on the list `StudID, CrsCode, Semester`.

Another example, the query

```
select T.CrsCode, T.Semester, T.Grade
from   Transcript T
where  StudID = '123456789'
```

can be dealt with effectively using a hash index on `StudID`. However, the B^+-tree index on `CrsCode, Semester` will not be helpful.