Exercise 1. Disk access and Indexing

DBMS execute the following query:

SELECT \* FROM emp WHERE salary = 200;

Suppose you have a B-tree index on "emp.salary". Also, assume that the table has 100 million tuples stored on 1 million pages on disk (i.e., an average of 100 tuples per page). Further assume that you have an infinite amount of main memory available, and that the entire B-tree is originally resident in the main memory, while no page of the table is in the main memory. Assume that random access to the disk takes 28 milliseconds while sequential access to the disk takes only 0.28 milliseconds.

**A)** Determine the cost (as total hard disk access time) to answer this query when you use the B-tree to answer the query. To do this, specify a formula that receives the number of tuples that have a content of 200 as the parameter.

Время чтения будет зависеть от n – числа искомых записей, числа страниц на диске = 1000000, времени рандомного доступа к диску = 28 мс и кластеринг фактора, который может лежать в отрезке [ceiling(n/100); min(1000000, n)]. Так как про кластеринг фактор мы ничего не знаем, будем рассматривать худший случай.

Предположим, что мы имеем худший случай кластеринг фактора, то есть записи на диске не упорядочены соответственно индексам. Тогда, так как мы читаем с помощью индекса, нам надо будет совершить максимальное число переходов между блоками на диске. Число этих переходов будет равно минимуму из числа искомых объектов – n и числа страниц на диске (миллион), так как, если у n < 1000000, то в худшем случае на каждой из n страниц будет ровно 1 нужный объект и придется читать n страниц, либо если n >= 1000000, все страницы будут содержать минимум по одному нужному объекту, и нам надо будет прочитать их все (учитывая, что оперативная память безграничного размера, при прочтении каждая страница будет кешироваться и вновь за ней идти на диск не надо будет). Таким образом финальная формула для худшего случая

Time = raTime\* min(1000000, n) = 28\* min(1000000, n) миллисекунд

В случае если у нас более отсортированная таблица с записями, время будет меньше.

**B)** For which parameter values (number of employees with a content of 200) is the use of the B-tree would be advantageous compared to a full table scan? How much better would be the full table scan than the index scan if all 100 million employees have a salary of 200?

Время на фулл скан таблицы без индексов будет вычисляться как произведение числа страниц на время последовательного доступа к диску, то есть 0.28 мс \* 1000000 = 280000 мс.

Сравниваем это значение с значением Time из прошлого пункта. Использование индексов будет выгодно тогда, когда

280000 – Time > 0

280000 - 28\* min(1000000, n) > 0

10000 – min(1000000, n) > 0

10000 > n

Таким образом, если искомых записей меньше 10000, выгоднее использовать индекс.

Если все записи искомые, то использовать фулл скан выгоднее. Посчитаем, насколько.

280000 – 28\*1000000 = -27720000 мс

То есть при фулл скане мы выиграем 27720000 мс. Время при фулл скане будет в 100 раз меньше времени при индексном доступе.

Exercise 2. DBMin

 For each of the following operators specify maximum size of main memory buffer it can utilize and minimal possible size of main memory buffer, both depending on the size of the input.

* Nested-loop Join

В случае минимального размера буфера мы можем сравнивать одну страницу из первой таблицы с каждой из второй по очереди до конца второй таблицы, затем заменять страницу из первой таблицы. Для этого понадобится буфер на две страницы. Стратегия замещения, наверное, MRU, потому что после обработки страницы из второй таблицы снова с ней работать алгоритм будет уже только на следующей итерации цикла по первой таблице. При этом максимальным размером буфера будет суммарный размер двух таблиц.

* (Grace) Hash Join

В случае хэш джойна стоит положить первую таблицу в буфер целиком как хэш таблицу, и далее перебирать страницы из второй таблицы. То есть минимальный размер буфера – размер первой таблицы +1. Максимально можно загрузить обе таблицы в буфер. В качестве стратегии замещения можно было бы использовать MRU, как и в прошлом случае.

* Sort Merge Join

При сортировке слиянием минимально требуется две ячейки буфера для попарного сравнения элементов. Максимально, наверное, хотелось бы поместить в буфер обе таблицы, подвергаемые слиянию. При этом стратегию замещения лучше выбрать вероятно LRU, потому что наиболее недавние страницы мы более вероятно будем сравнивать в скором времени.

* Table Scan

При считывании и обработке таблицы целиком буфер нужен минимального размера для 1-2 обрабатываемых страниц. В теории его максимальный размер может быть равен количеству обрабатываемых страниц, если мы по какой-то причине решим поместить их все в память. При этом наиболее оптимальной стратегией замещения для считывания таблицы считается MRU, так как после обработки страницы она нам уже не понадобится.

* Index Scan (Access to a table using a B-tree. Caution: you need two buffers: one for the B-tree and one for the table blocks)

При случайном доступе по индексу становится практически не важной стратегия замещения, потому что сложно предположить, какая страница нам может понадобиться следующей. Буфер для таблицы также может быть любого размера, минимум – 1 страница, максимум – размер таблицы. Для самого би-дерева имеет смысл хранить несколько самых частоиспользуемых уровней дерева, то есть несколько верхних уровней, как минимум 1 – его корень, максимум можно положить в буфер все дерево целиком. При этом лучше всего для хранения дерева использовать стратегию замещения FILO, чтобы корень по возможности никогда не уходил из буфера.

Which replacement strategy would you use?