Implementation of Databases

Assignment 4

Participants:
(sorted in last name order)
Ulfet CETIN
Shreya KAR
Samuel ROY

Exercise 4.1 (Cost Estimation)

1. Number of tuples retrieved = Cardinality* reduction factor

   (a) Employee :
   Cardinality=50,000
   conditions in where clauses e.sal>59,0000 and e.hobby='yodeling'
   reduction factor for e.sal>59000 (note index is on column sal)
   = High(I)-value/High(I)-Low(I)
   =(60,000-59,000)/(60,000-10,000)
   =1/50
   reduction factor for e.hobby='yodeling'
   =1/(Nkeys)
   =1/200
   Therefore, total tuples= 50,000*1/50*1/200 =5

   (b) Dept: Cardinality=5000
   conditions in where clauses d.floor=1
   reduction factor for d.floor=1
   =1/(Nkeys)
   =1/2
   Therefore, total tuples= 5000*1/2 =2500

   (c) Finance : Cardinality=5000
   No where clauses applicable for this relation individually
   Therefore, total tuples= 5000

2. Sequence of join for the query plan: $((Employee \bowtie Department)) \bowtie Finance))$
   Step 1 : Apply selection conditions to Employee
   Since a unclustered B+ tree index is built on E.sal,
   total number of tuples selected where e.sal>59000
   =50000/50
   =1000
   total cost of retrieving these 1000 tuples
   =cost of traversal to leave node + scanning leave nodes+cost of retrieving those selected tuples
   = 3+ (1000/200)+ 1000
   =3+5+1000
   =1008
   Step 2: Now applying the selection condition e.hobby='yodelling' on the fly to 1000 selected tuples
   number of selected tuples
   =(1000/200)
   =5
   Step 3: Join Employee and Department
   This will be index nested loop join, therefore cost of join
   = number of tuples of outer relation * (cost of tree traversal + I/O cost)
   =5*(3+1)
   =20
   Step 4: select tuples where d.floor=1
   (Note : only 5 tuples will be retrieved as we need one corresponding tuple from department for every one of the 5 selected tuples from employee)
   =(5/2)
   =3
   Step 5: Join Department with Finance
   This is a also a nested loop join,therefore cost of join
   = 3*(3+1)
   =12
   Step 5:
   Final cost
   = 1008+20+12
   =1040

Exercise 4.2 (Physical Database Design)

Given a relational table Student(sno,name,age,marstat) which is stored in an unsorted heap file with 1,000 pages (primary key is sno). Your system should be optimized for the following queries:

1. Q1: SELECT * FROM Student WHERE sno = 534558
2. Q2: SELECT name,age FROM Student WHERE age > 19 AND age < 24

How do you physically organize your database? Which indexes (clustered/unclustered) should be created to optimize the overall performance for both queries? (For the fan-out of tree index G we take 100.)

→ the physical organization of database

First of all, using an unsorted heap file does not cater to what we need, given the two queries above. Looking at the queries,

- query 1 aims to find a specific sno of a student (sno is the primary key, at most one record exists with that sno)
- query 2 aims to fetch an age range (assuming a good selectivity factor, we will fetch not single but some records)

Based on this realization, the first step would be arranging the relational table in a way that it will be sorted on *age* attribute.

We are not told anything about the the updates(deletion, update, insertions) so we do not take their burden on indexes and file structure in consideration.

How it will come to our help will be clear in the explanation of how we selected the indexes.

→ index selection to optimize general performance

- For the first query, building **an unclustered hash table on *sno* field** would help us greatly.

  Hash table, even if it is multi-leveled, would require in magnitude of single digits to find which page the specific record is stored. As we would fetch single record, hash table would grant us the record we want with all the fields it has(as SELECT * indicates) in I/O of single digits.

- For the second query, building **a clustered B+ tree on *age* field** would help us greatly.

  As we noted in the "physical organization of database" part, it is better that we sort the file on age field. Assuming we did so, B+ tree would allow for us to reach the first record that has the age of 19.

  From this record on, age field of the records will not decrease; this field will either stay the same or increase.

  Good thing about a clustered B+ tree on a field is, finding the ranges are so efficient on them because:

  1) B+ tree allows to reach the first record(or its page number) in very few number of I/O operations, allowing skipping of a lot of pages.
  2) Index being clustered means the actual file that the records are stored in are sorted on the search key. This allows starting from the lower end of the range, scanning sequentially up until higher end is exceeded by the scanned tuples' field value.

→ notes on the approach

- Note that in question body, we are not given any concerns about the update/delete/insert operations. Thus, we did not care about the frequency of those said operations and assumed that only queries that has the highest and considerable frequency are those two given in the question body.
- We were not told whether we are allowed to rearrange the file that the records are kept. Yet, the first part of the question involves physical database design. Therefore, we assumed we are allowed to, and we made our decisions based on this.

Exercise 4.3 (Cost Estimation)

Consider a relation R(a, b, c, d) containing 5,000,000 records, where each data page of the relation holds 10 records.

1. Suppose R is organized as a sorted with indexes, and R is stored in R:a order. There are three access paths:
   A1. Access the sorted for R directly.
   A2. Use a clustered B+ tree index on attribute R:a.
   A3. Use a unclustered hash index on attribute R:a.

   For each of the following selection queries, state which of the three access paths is most likely to be the cheapest and explain why.

   (a) $\sigma_{a=50000}$ (R)

   (b) $\sigma_{a\neq 50000}$ (R)

   (c) $\sigma_{a>50000} \wedge \sigma_{a<50010}$ (R)

   (a) $\sigma_{a=50000}$ (R)- A unclustered Hashed index is the cheapest here. Retrieving data entries cost D and to fetch the data record from the file which cost another D. The total cost is 2D, therefore even lower than cost of tree index.

   (b) $\sigma_{a\neq 50000}$ (R)- Since the selection will require a scan of the available entries and we are starting at the beginning of the sorted index, accessing the sorted file should be slightly more cost efficient,because of lookup time.

   (c) $\sigma_{a>50000} \wedge \sigma_{a<50010}$ (R)- A B-Tree is the cheapest here.the first record that satisfies the selection is located, Subsequently, data pages are sequentially retrieved (using the next and previous links at the leaf level) until a record is found that does not satisfy the range selection;

2. Assume that all four attributes of R are string fields of the same length. There are 1000 buffer pages. A projection query $\Pi_{a,b}(R)$ should be executed. 20 records of the resulting relation can be stored in one page. Consider the optimized version of the sorting-based projection algorithm: The initial sorting pass reads the input relation and creates sorted runs of tuples containing only attributes a and b. Subsequent merging passes eliminate duplicates while merging the initial runs to obtain a single sorted result (as opposed to doing a separate pass to eliminate duplicates from a sorted result containing duplicates).

   (a) How many sorted runs are produced in the first pass? What is the average length of these runs? (Assume that memory is utilized well and any available optimization to increase run size is used.) What is the I/O cost of this pass (**including** writing of the intermediate result)?

      1) How many sorted runs are produced in the first pass?

         According to book[1], we can write out 2*B internally sorted pages, where B is the number of buffer pages, with an aggressive approach, which means that a run in the first step can consist of 2*B pages.

         # pages that the tuples that are cleared of unwanted fields occupy: 5.000.000 / 20 = 250.000
         (as said in the question body, each buffer block can store 20 resulting tuple, hence the division by twenty)

         # pages each run consists of = 2*B = 2*1000 = 2000

         # sorted runs = $\lceil 250.000 \ / \ 2000 \rceil = 125$

      2) What is the average length of these runs?

         As noted above, it is 2000 pages.

      3) What is the I/O cost of this pass (**including** writing of the intermediate result)?

I/O Costs	=	Read-in cost of the relation	+	Write-back cost of the resulting tuples		
I/O Costs	=	500.000 pages read	+	250.000 pages write	=	750.000 I/Os

   (b) How many additional merge passes are required to compute the final result of the projection query? What is the I/O cost of these additional passes? Writing of final result is **excluded**.

      1) How many additional merge passes are required to compute the final result of the projection query?

         Pass 1 # resulting runs = $\lceil 125 \ / \ 2*1000 \ \rceil = 1$
         As we reached the single run, we are ready to output the result, the resulting tuples are all sorted. No extra pass is necessary.

         1 additional merge pass is required.

      2) What is the I/O cost of these additional passes?

         The runs that was written back in the first run has to be read in this 1 extra pass (it is told that writing of the final result is excluded).

         There are 250.000 pages that the resulting tuples occupy, consisting of 125 sorted runs. All has to be read in order to be merged.

         I/O cost of these additional pass (excluding writing of final result) = 250.000 I/Os

[1] Ramakrishnan, R., & Gehrke, J. Database Management Systems, 2^{nd} Edition.

3. Suppose the following query with a self-join on R is executed:$\Pi_{a,b}(R \bowtie_{a=a} R)$. Note that a is not a key attribute. Which join method (block-nested loop join, hash join, or sort-merge join) using any of the access paths from above is most likely to perform best. Why? What are the estimated costs?

   M = 500000 pages
   B = 1000 buffers

   Block nested loop join:
   Formula: M+(ceil(M/(B-2))*N)
   500000 + (ceil(500000/998) * 500000) = 251500000

   Hash join:
   Formula: 3(M+N)
   3 * (500000 + 500000) = 3000000

   Sort-merge join:
   Formula: O(M log M)+O(N log N)+M+N
   2 * 2 * 500000 + 2 * 2 * 500000 + 500000 + 500000 = 5000000

   Suppose we have 1000 buffers. So, we need 2 passes to merge the results, and we are doing read and write for all the passes. As per the calculation Hash Join is the cheapest method.

Exercise 4.4 (Answer questions briefly)

1. Reason below formulas in detail

   (a) Why is the cost for an equality selection using a clustered tree index D * (logG 0:15B)?

   (b) Why is the cost for an insert operation using an unclustered hash index 4D?

   Base your explanation on the below assumptions from empirical studies:

   - In a sorted, pages are stored sequentially, retrieving a desired page directly only needs one disk I/O.
   - In a clustered , pages are usually 67% full, and the number of physical data pages is 1.5B.
   - We omit the time for processing a record in memory (since it is usually negligible compared with the time for reading or writing disk pages)
   - D is the cost for one I/O operation.

   (a) Once the equality selection matches the search key, the first page containing the desired record is located in logG(1.5B) steps,that is by fetching all pages from the root to the appropriate leaf. Each step requires a disk I/O and two comparisons.The cost is D*logG(1.5B) where D is average time to read or write a disk page and G is the fan-out.

   (b) The record is first inserted in heap file,at a cost of 2D.In addition, the appropriate page in the index must be located,modified to insert a new data entry, and then written back. The additional cost in writing back is 2D. So totally it is 4D.

2. Explain the CAP Theorem.
   CAP Theorem states that a distributed computer system cannot concurrently provide more than two of the following three assurances all at optimal levels.:

   - Consistency : Consistency refers to every client/every node in a distributed cluster having the same view of the data.
   - Availibility: Every non-failing node returns a response for all read and write requests in a reasonable amount of time.
   - Partition tolerance: The system continues to function and upholds its consistency guarantees in spite of network partitions.

3. What is the default join method in Spark? Explain how it works.
   The default process of join in apache Spark is called a shuffled Hash join. The shuffled Hash join ensures that data on each partition has the same keys by partitioning the second dataset with the same default partitioner as the first. First, it Map through two different data frames/ tables. Uses the fields in the join condition as the output key. Shuffle both datasets by the output key. In the reduce phase, join the two datasets now any row of both tables with the same keys are on the same machine and are sorted.