

Problem Set 2: Query Optimization

The purpose of this problem set is to teach you about the process of query optimization, including at both the logical (e.g., algebraic) and physical (e.g., access method and operator implementation) levels.

Part 1 - Warmup

- We studied three disk based 'access methods' in class – heap files, B+trees, and hash files built via extensible hashing. Suppose you have a table A with $|A|$ records in it and a table B with B records in it, where $|A| < |B|$. Each table has fields, f_1 and f_2 . For each of the following query workloads, describe which access methods would be best suited to that workload, including whether any indices should be clustered (primary) or unclustered (secondary). You may assume that sequential disk I/O is ten-times more efficient than random I/O. Justify each answer with a short sentence or analytical argument. You may disregard insertion and deletion costs.
 - A sequential scan of all elements of $|A|$.
 - A join of $|A|$ and $|B|$ of the form $(A.f_1 = B.f_1)$.
 - 50% range queries of the form $(A.f_1 > k)$ on A (where k is a constant), and 50% joins of the form $(A.f_1 = B.f_1)$.
 - 90% range queries of the form $(A.f_1 > k)$ on A , and 10% joins of the form $(A.f_2 = B.f_2)$.
 - 33% joins of the form $A.f_1 > B.f_1$, 33% sequential scans of B , and 33% lookups of the form $B.f_2 = k$.

Part 2

Bob is a vacuum cleaner salesman. He has decided to store information about his clients and sales in a database. He wants to keep the following information:

- For each customer, he wants to store his or her name, address, and the sales price of all the vacuums he or she has purchased (some customers own multiple vacuums, and one crazy old man even own three of the same type of vacuum!) Bob sells his vacuums for different prices to different customers.
- Each vacuum model has a different horsepower rating which Bob also wants to store.
- As a way to increase his profit margin, Bob tries to get customers to buy a one-year 'service contract' with the vacuums he sells. Bob visits customers who have bought a service contract and maintains their vacuum for them. Thus, for each sale, Bob wants to keep track of the sales price and the type of service contract (if any) attached to the sale.
- Each type of service contract has a service frequency associated with it, which Bob wants to store in the database.
- For all service contracts, Bob wants to store the last time the customer's vacuum was serviced.

Bob, knowing he's not being the brightest bulb on the tree, is concerned that he won't be able to write complex queries that combine readings from multiple tables together to get information about his customers and their vacuums. Thus, he proposes to store his database in a single table, reasoning that this way he only has to store one record per vacuum he sells, and it will be very easy for him to lookup information about his business. His schema is as follows:

```
vacuums: {cid | name | city | street | no | model_no | hp | price | contract_type | svc_freq | last_svc }
```

This representation works fine for a few months, but as Bob's customer base grows, he finds it is very hard to maintain this database, particularly as customer's retire their old vacuums in favor of new models with new service contracts. He hires you to help him fix his database system.

- Derive a set of non-trivial functional dependencies for this database from Bob's requirements given above.
- List three update, insertion, or deletion anomalies that Bob might encounter using the schema he has proposed.

4. Use your functional dependencies to derive a schema in BCNF for Bob's database.

Does your schema preserve all of the functional dependencies you began with? Does it address all of the anomalies you listed above?

5. Suppose that Bob wants to find the name's and addresses of all of the customer's who live on Main St. whose vacuums are due for service. Suppose that 75% of customers own service contracts, 25% of them live on Main St, and 10% of the service contracts (irrespective of contract type) are due for service. Assume there is no relationship between where a customer lives, the likelihood that he or she purchases a service contract, or the likelihood that a particular contract is due for service. Write a SQL query that accomplishes this for both Bob's original schema and for the schema you designed. Show equivalent relational algebra operations for these two queries, and argue that the algebra you have selected is an efficient way to execute your query given the operator selectivities above.

6. For each of the operators in your two query plans, describe the particular implementation (i.e., physical plan) you would choose given the parameters below. You should consider issues such as the appropriate choice of join algorithm, which relation is inner or outer, and so on. You should compute the approximate number of I/Os for each of the two plans.

Assume that all of your tables are stored in a heap file with no indices. You may also need to know that Bob has sold 10,000,000 vacuums to 8,000,000 customers under 10 different types of contracts. Assume that names, cities, and streets each require 20 bytes to store, and that all other record types (including null values) require 2 bytes to store. You may assume that there is no additional per-tuple storage overhead. Bob's computer has 1MB of memory for query processing, and a disk page is 10KB.

7. Suppose that you are allowed to add primary and/or secondary indices to the tables in your schema (you do not need to consider Bob's schema in this question.) Describe what indices you would add, and recompute the overall number of I/Os when these indices are available. You may assume that selections and join predicates can be pushed into access methods via SARGs (e.g., that if you have a hash file on street name, you can directly look up customers who live on Main St.)

Part 3

8. In a question unrelated to Bob's vacuum business, suppose that you are designing a database system that stores each column of data in a table in a separate file on disk. Such a representation can be efficient when, for example, most queries access only a few columns but each table consists of tens or hundreds of columns, since each query must then access only the disk pages corresponding to columns it references. When a representation of this sort is used, the database system needs a facility figure out which records of a given column correspond to records in another column. If all columns are stored in the same order, this correspondence is trivial; however, we may wish to store different columns in different orders to facilitate common queries over columns in a particular order. For example, if we store a salary column in sorted order, it is very easy to look up employees who make more than \$50,000.

In the situation where columns are stored in different sort orders, we can store a special column, $P_{A \rightarrow B}$, of values called a "permutation", that maps the i th entry in a column A to the $P_{A \rightarrow B}(i)$ th entry in B . With $n - 1$ (where n is the number of columns in the original table) such permutations, we can reconstruct the original table.

Your job is to devise an efficient external (i.e., disk-based) algorithm for permuting a column A into the order of a column B using a permutation $P_{A \rightarrow B}$. Assume that you have M bytes of memory, and that A and P each require n bytes to store and consist of r records. Assume that $n \gg M$ and that a disk pages are B bytes. Characterize the performance of your algorithm in terms of the number of sequential and random disk I/Os (i.e., page reads and page writes) it requires.

An example of the permute operation is shown below:

```
permute(A={a,b,c,d,e,f,g}, P_{A \rightarrow B}={2,1,4,5,6,3,7}) =
P[1] = 2; A[2] = b
P[2] = 1; A[1] = a
P[3] = 4; A[4] = d
P[4] = 5; A[5] = e
P[5] = 6; A[6] = f
P[6] = 3; A[3] = c
P[7] = 7; A[7] = g
=> b,a,d,e,f,c,g
```