Query Optimization

Note: you may find Chapter 15: Query Processing in the textbook to be rather helpful in this assignment.

1 Question 1

A database stores warehouses and products. Here is the schema:

```
CREATE TABLE warehouse(
    name VARCHAR(100) PRIMARY KEY,
    zip_code VARCHAR(20) NOT NULL
);

CREATE TABLE product(
    id int PRIMARY KEY,
    description VARCHAR(600) NOT NULL
    availableAt VARCHAR(100),
    FOREIGN KEY (availableAt) REFERENCES warehouse(name)
);
```

Here are some facts about the data: - There are 500,000 products and 10,000 warehouses in the database. - The average size of a warehouse record is 50 bytes. - The average size of a product record is 100 bytes. - There are 500 known zip codes, and warehouses are uniformly distributed across zip codes. - Scanning through a table takes 0.1 ms per block. - The disk block size is 8K = 8192 bytes. - Buffer size is 4 blocks.

You want to find all products stored in Providence warehouses at the zip code "02904-2413". For each of the following query plans, find the **approximate** runtime. Then, discuss which plan should be selected and why: 1. Select `warehouses` (02904-2413), and then join with `product` using an block nested loop join. 2. Select `warehouses` (02904-2413), and then join with `product` using a sort-merge join. 3. Join `product` with `warehouse` using a block nested loop and then select the merchandise in (02904-2413).

1.1 Solution

Note: the point of this question is to practice rough approximations based on your understanding of the algorithm. If you are off by a few constants, that's perfectly fine, so long as the process demonstrates that you kept the algorithm in mind when computing your answer.

For all of the following, we note that there are about 10,000 / 500 = 20 warehouses at any given zip code. This will take up 20 * 50 = 1000 bytes which is less than a block, and so we can keep all of the relevant warehouses in memory in a single block. We also note that the products table takes up 500,000 * 100 / 8192 = 6104 blocks, and the warehouses table takes up 10,000 * 50 / 8192 = 61 blocks.

1. Select Providence warehouses (02904-2413), and then join with product using an block nested loop join.

First, we do a table scan and then retrieve all of the warehouses at zip code 02904-2413. We have 61 blocks to loop over, which takes 6.1ms. Then, we keep all of these warehouses in memory and scan through all 6104 product blocks, which takes 610.4ms. In total, this takes 616.6ms.

2. Select Providence warehouses (02904-2413), and then join with Product using a sort-merge join.

First, we do a table scan and then retrieve all of the warehouses at zip code 02904-2413. We have 61 blocks to loop over, which takes 6.1ms. We sort the warehouses by `name` which can be done in memory. Then, we sort the product table on the availableAt key, which will require 6104 (2 ceil(log_3(6104/4)) + 1) = approx 91,560 block accesses. This takes 9,156ms

Then, we do a merge join, which requires reading 1 + 6104 blocks, which takes 610.5ms. In total, about 9772.6ms.

3. Join Product with Warehouse using a block nested loop and then select the merchandise in (02904-2413).

First, we do a block-index nested loop with the warehouse table as the outer table, keeping 3 warehouse blocks in memory at a time. This requires 61 + 61 * 6104 / 3 = 124,135 block transfers. This takes 12,413.5ms.

Next, we do a select. We know we should have at most 500,000 rows, meaning 6104 blocks, meaning another 610.4ms to scan and select. In total, about 13,023.9ms.

2 Question 2

Consider the relations r1(A, B, C), r2(C, D, E), and r3(E, F), with primary keys A, C, and E, respectively. Assume that r1 has 1000 tuples, r2 has 1500 tuples, and r3 has 750 tuples. Estimate the size of $r1 \bowtie r2 \bowtie r3$, and give an efficient strategy for computing the join. Hint: your strategy should involve the creation of indices.

2.1 Solution

The relation resulting from the join of r1, r2, and r3 will be the same no matter which way we join them, due to the associative and commutative properties of joins. r1 joined on r2 will yield a relation of at most 1000 tuples, sine C is a key for r2. Likewise, joining that result with r3 will yield a relation of at most 1000 tuples beause E is a key for r3. Therefore, the final relation will have at most 1000 tuples.

An effient strategy for computing this join would be to create an index on attribute C for relation r2 and on E for r3. Then for each tuple in r1, we do the following: a. Use the index for r2 to look up at most one tuple which matches the C value of r1. b. Use the created index on E to look up in r3 at most one tuple which mathes the unique value for E in r2.

3 Question 3

Consider a relation r1(A, C) and r2(A, B), under what conditions are the following queries equivalent? Hint: It might help to convert this to relational algebra first. Hint: Think about the size of r1 joined on r2.

```
SELECT A, B, SUM(C)
    FROM r1 JOIN r2
    GROUP BY A, B;

SELECT A, B, C
    FROM r2 JOIN
    (SELECT A, SUM(C) as C
        FROM r1
        GROUP BY A);
```

3.1 Solution

They will be the same no tuple in r2 has a non-unique A value that matches some A value in r1. This is because If A is unique in r2, then the aggregation in both will be the same no matter if we took it with r2 and r1, or just r1. Since the aggregations are the same, then the join overall will be the same. If A is not unique in r2, then the aggregates will be different, since joining on non-unique keys can lead to duplicity of the aggregate key, which in the case of `SUM`, changes the answer. An interesting note to make is that if `SUM` were replaced by either `MIN` or `MAX`, these two joins will always be the same.

4 Question 4

Given the following relational schema, SQL query, and DB statistics, calculate the approximate number of block transfers for each of the following scenarios.

Schema:

```
sells (product_id, store_id, price, volume)
```

SQL Query:

```
SELECT product_id
    FROM sells
    WHERE (store_id=42
        AND volume < 32);
```

DB Statistics: - Number of files the relation is contained in: 1 - Number of tuples in the relation: 10M - Number of disk blocks containing the relation tuples: 100K - Distribution of the store_id attribute : Uniform (min=0, max=999) - Distribution of the volume attribute: Normal ($\mu = 50$, $\sigma = 10$)

1. There is no index on the database.
2. There is a primary, sparse B+ tree index on the attribute Sells.store_id, and the cost of traversing this index is b1.
3. There is a primary, sparse B+ tree index the attribute Sells.volume, and the cost of traversing this index is b2.
4. There is a secondary, dense B+ tree index on the attribute Sells.store_id, and the cost of traversing this index is b3.
5. There is a secondary, dense B+ tree index on the attribute Sells.volume, and the cost of traversing this index is b4.

4.1 Solution

1. There is no index on the database.

Since there is no index, and we assume that the data is not sorted in any way, every single block will have to be read. Thus, there will be 100,000 block transfers.

2. There is a primary, sparse B+ tree index on the attribute Sells.store_id, and the cost of traversing this index is b1.

Since we're looking for attributes that fulfill `store_id=42`, we can just traverse the tree once to get pointers to the blocks that meet this requirement, and read them all in order (since this is a primary, sparse tree, though, we'll likely just find the first one in memory then do sequential access).

We have a uniform distribution of the store_id attribute, which means we can expect about 10,000,000 / 1000 = 10,000 tuples to meet this requirement. Since we have 100,000 blocks, this amounts to about 10,000,000 / 100,000 = 100 entries per block. Therefore, we have to read about 10,000 tuples / 100 tuples/block = 100 blocks. Adding our original cost, this is 100 block reads + b1. Then processing is done to eliminate tuples that fail the other field.

3. There is a primary, sparse B+ tree index the attribute Sells.volume, and the cost of traversing this index is b2.

With some quick maths, we find that with mean 50 and standard deviation 10, we have that around 3.59% of our entries will be less than 32. Therefore. 10,000,000 * 3.59% = 359,000 tuples. This amounts to 3590 blocks to be read (see answer to 4.2). Since we're on a primary index, we just need to traverse to the first element, and then scan across the leaf nodes for all of the matching values. To formalze, our total cost is 3590 block reads + b2.

4. There is a secondary, dense B+ tree index on the attribute Sells.store_id, and the cost of traversing this index is b3.

From 4.2, we know that we can expect around 10,000 tuples to meet the requirement that `store_id=42`, and since this is a dense B+Tree, each one of these will have an entry in a leaf node. Therefore, we can traverse the tree once to get the left-most that fulfills this requirement, then just traverse the leaf nodes in order. However, since this is a secondary B+Tree, we don't know that the tuples will be in sequential order on disk, i.e., they could be in different blocks. Since we have 100,000 blocks, which is greater than 10,000, we have, in the worst case, 10,000 block accesses plus our initial traversal. To formalize, our cost is 10,000 block reads + b3.

5. There is a secondary, dense B+ tree index on the attribute Sells.volume, and the cost of traversing this index is b4.

From Scenario C, we know that we have around 359,000 tuples that meet the requirement that `volume<32`. We can do the same thing as in Scenario C by finding the left-most, then traversing leaf nodes until this condition is no longer met. However, since this is a secondary B+Tree, we don't know that the tuples will be in sequential order on disk, i.e., they could be in different blocks. Since we have 100,000 blocks, which is less than 359,000, we have, in the worst case, 100,000 block accesses plus our initial traversal. To formalize, our cost is 100,000 block reads + b4.

Feedback

As this is a new course, we appreciate any feedback you have for us! If you enjoyed this assignment, hated this assignment, or have other thoughts to share, please do so here!