TU Wien

Data Science

184.780 Advanced Database Systems

Exercise 1

Mal Kurteshi 11924480

March 19, 2021

1 Introduction

In this report i am presenting my solution regarding the first assignment in the course *184.780 Advanced Database Systems*. The assignment consists on 4 exercises.

2 Exercise 1: Disk access in database

2.1 Sequential Scan

2.1.1 Disk A - Magnetic disk

In order to calculate the I/0 for the magnetic disk we are based in the formulas below:

$$T_a = T_aFlight + T_aAirline$$

where we have

$$T_a = T_s + T_r + Bck_{no} * T_{tr} + Tck_{no} * T_{t2t}$$

We need to be clear that the access time will be calculated for both $Flight$ and $Airline$ files but before that let us calculate some of the hardware specification results.

Some of the parameters we already know like *average seek time* T_s which is **6 ms**. The *average rotational delay* T_r is **2.5 ms** calculated form *average time of delay * (1/rotational speed) * convert to ms*, which in numeric parameters takes the form.

$$T_r = 0.5(1/12000) * 60000 = 2.5ms$$

The *transfer time* T_{tr} for a block of **8kb** is **0.04 ms** calculated from the formula $\frac{blocksize}{transferrate}$ presented in numbers $\frac{8kb}{200000kbs}$. And the *text to text seek time* is given to us and is **3ms**.

Now we need to calculate the two parameters Bck_{nr} and the Tck_{nr}. Let us first focus on the Flight file. In the Flight file we have 280000 records of 10300 bytes. Considering the size of the file in terms of records and bytes which needs to be stored in disk and also the disk size of $8kb$, we need can calculate the size in bytes that we need to store which is $280000 * 10300 = 2884000000bytes$. We can assume that every block if filled and no empty spaces we have $\frac{2884000000}{8000} = 360500$ representing the Bck_{nr} blocks in which we can stoure the data. The other parameter to be calculated still is the number of tracks Tck_{nr} which is calculated as $\frac{filesize}{blocks} inatrack$ in our case we have $\frac{360500}{68} = 5302$ Since we have all the parameters that we need to be able to calculate the *access time* T_a for the Flight file.

$$T_aFlight = 6ms + 2.5ms + 0.04ms * 360500 + 5301 * 3ms = 30331.5ms$$

Similarly with the Flight case we also calculate the Airline case. In here we have different parameters but the procedure is the same. Giving us the results below:

$$T_aAirline = 6ms + 2.5ms + 0.04ms * 27 + 1 * 3ms = 12.08ms$$

Now the total *access time* T_a is the sum in between two components below for the particular files.

$$T_a = T_aFlight + T_aAirline = 30331.5ms + 12.08ms = 30343.58ms$$

2.1.2 Disk B - SSD

In this section we are working with SSD disk, in order to calculate the total time we are beeing based in the following equations:

$$T_a = T_aFlight + T_aAirline$$

where we have

$$T_a = (Bck_{no} * \text{organization block size/disk block size}) + \text{access time}$$

In difference to Disk A, disk B has a block size of **5MB** and the transfer rate is **400Mb/s**. Having problems with defining Bck_{no}!

2.2 Index Scan

3 Exercise 2 : Selectivity

3.1 a

We know that we have a table exam with 38000 rows. Moreover, we know that the DBMS uses an equi-depth histogram to support faster selectivity calculation for query planning. As it is specified, the histogram divides the column values into 6 groups: 115, 210, 370, 420, 500, and store the maximum value in a column 1200. This means that every group contains 6333,33 values.

3.1.1 pages $>$ 190

In this case we have the groups distribution 115-210, 210-370, 420-500, 500-1200, we are looking for the selectivity for pages $\geq$ 190. So we are collecting all the groups after 115-210 which contains 4 groups with 6333,33 values = 25333,3. As for the group 115-210 we need to split it since it contains the value 190, we have here in 115-210, 95 range values if we take it in percentage from 190 its 21 percent in the group, so with some calculations 21 percent of 6333,33 is 1330. At the end we have: 25333,3 + 1330 = 26663

$$\text{book}(\text{page} \leq 190) = 26663/38000$$

3.1.2 pages > 520

Similar to the case above here we have a different condition of pages > 520. So we have the group 500-1200 containing 6333,3 values. In the group 500-1200 we need to select the values grater than 520, so this means selecting 679 values in this range if we translate it as a percentage we come up with 97 percent values should be included from this group in the results. Leading us to the result of 6143 values out of 38000

$$\text{book}(\text{page} > 520) = 6143/38000$$

3.2 b

3.2.1 pages = 143

In this section we have the case with constraint pages = 143, at this point we can make an assumption for two possibilities that page either is unique or not. If it is a unique value we know that the selectivity is 1. In the other hand in the case where it is not unique value we do not know the selectivity for sure but we can define the maximum and minimum boundary, for the minimum boundary we know for sure it should be higher than one so minimally 2. As for the upper bound we can take into consideration the case were where all the values are equally distributed and would have a selectivity of 52. Which we can see also below:

$$\text{book}(\text{page} = 143) = 1/38000$$

$$\text{book}(\text{page} = 143) = 2/38000$$

or

$$\text{book}(\text{page} = 143) = 52/38000$$

3.2.2 pages != 870

If we have an inequality constraint such as pages != 870 we can assume that every value in the range not equal to 870 would fulfill the constraint. In our case we have 730 unique values so we assume that there will remain 729 values which satisfy this condition. And since we have 38000/730 = 52 different values, the percentage that satisfies our condition is as follows: 729 * 52 = 37908 values from 38000.

$$\text{book}(\text{page}! = 870) = 37908/38000$$

3.3 c

3.3.1 book $\bowtie_{onby=penname}$ *author*

The selectivity of equi-join via foreign key from R to S is already given as:

$$sel_{R.A=S.B} = \frac{1}{|S|}$$

Thus the selectivity of our relation is: $1/|500|$

3.3.2 $\Pi_{by}(book)$

The projection operation selects the columns of a table. While it changes the structure of tuples returned, it does not do anything to limit the percentage of the tuples returned. Thus:

$$sel_{\Pi_{by}(book)} = \frac{38000}{38000}$$

3.4 d

I would suggest the first join to be JOIN edition e ON e.of = b.id because of the primary key involved, this could optimize the process seeing that the first join JOIN author a ON a.penname = b.by is not joining attributes on unique primary key but on an not unique once. Also i would opsition the operator $a.bookswritten > 5$ before the $ANDb.pages < 190$ because i assume that it would optimise this linkage in between these two joins.

3.5 e

Taking into consideration the fact that the joins performed there are equi-joins,i would suggest that the best strategy for the query above are the hash join or hybrid hash joins. Hash joins because of its efficiency regarding other join strategies. It is also important and good to know the primary keys of the tables and also to know whether the values in the fields that are used to join the tables are unique or not. Another important information is to know whether the data is ordered or not, as this would help with selecting the most efficient join strategy.

4 Exercise 3 : The Query Planner and You

First we have connected to the bordo.dbai.tuwien.ac.atAssignment by the credentials provided to us. After establishing a connection i have created the tables under triangleget.sql Then i executed the transacion for translation into SQL as it can be seen below:

```
u11924480=> SELECT a,b,c FROM r NATURAL JOIN s NATURAL JOIN t;
```

4.1 Which join strategy is chosen by default?

We can say that the join strategy is chosen by default but also we need to note that postgres chooses the join strategy based on the query planner. Thus it creates

estimations for the different join strategies and chooses the best one. In our particular case, the best estimated join strategy is the merge-sort join as it can be seen in the image below:

```
u11924480=> EXPLAIN SELECT a,b,c FROM r NATURAL JOIN s NATURAL JOIN t;
                                 QUERY PLAN
-----------------------------------------------------------------------------------
 Merge Join  (cost=51439.59..85289.62 rows=1983849 width=12)
   Merge Cond: ((t.a = r.a) AND (t.c = s.c))
   ->  Sort  (cost=1717.77..1767.77 rows=20000 width=8)
         Sort Key: t.a, t.c
         ->  Seq Scan on t  (cost=0.00..289.00 rows=20000 width=8)
   ->  Materialize  (cost=49678.66..51697.45 rows=403757 width=12)
         ->  Sort  (cost=49678.66..50688.05 rows=403757 width=12)
               Sort Key: r.a, s.c
               ->  Hash Join  (cost=54.00..5180.57 rows=403757 width=12)
                     Hash Cond: (r.b = s.b)
                     ->  Seq Scan on r  (cost=0.00..289.00 rows=20000 width=8)
                     ->  Hash  (cost=29.00..29.00 rows=2000 width=8)
                           ->  Seq Scan on s  (cost=0.00..29.00 rows=2000 width=8)
(13 rows)
```

4.2 Compare of performances in between join strategies

During the analyse in performance and the affect of different strategies the following outcomes have been recognised. We can see also from below that different join strategies can also lead to different number of rows.

```
u11924480=> EXPLAIN SELECT a,b,c FROM r NATURAL JOIN s NATURAL JOIN t;
                                 QUERY PLAN
-----------------------------------------------------------------------------
 Hash Join  (cost=13208.92..266716.41 rows=1983849 width=12)
   Hash Cond: ((t.a = r.a) AND (t.c = s.c))
   ->  Seq Scan on t  (cost=0.00..289.00 rows=20000 width=8)
   ->  Hash  (cost=5180.57..5180.57 rows=403757 width=12)
         ->  Hash Join  (cost=54.00..5180.57 rows=403757 width=12)
               Hash Cond: (r.b = s.b)
               ->  Seq Scan on r  (cost=0.00..289.00 rows=20000 width=8)
               ->  Hash  (cost=29.00..29.00 rows=2000 width=8)
                     ->  Seq Scan on s  (cost=0.00..29.00 rows=2000 width=8)
(9 rows)

u11924480=> set enable_hashjoin=0;
SET
u11924480=> set enable_mergejoin=1;
SET
u11924480=> EXPLAIN SELECT a,b,c FROM r NATURAL JOIN s NATURAL JOIN t;
                                    QUERY PLAN
-----------------------------------------------------------------------------------
 Merge Join  (cost=54174.93..88024.96 rows=1983849 width=12)
   Merge Cond: ((t.a = r.a) AND (t.c = s.c))
   ->  Sort  (cost=1717.77..1767.77 rows=20000 width=8)
         Sort Key: t.a, t.c
         ->  Seq Scan on t  (cost=0.00..289.00 rows=20000 width=8)
   ->  Materialize  (cost=52414.00..54432.79 rows=403757 width=12)
         ->  Sort  (cost=52414.00..53423.39 rows=403757 width=12)
               Sort Key: r.a, s.c
               ->  Merge Join  (cost=1856.44..7915.91 rows=403757 width=12)
                     Merge Cond: (s.b = r.b)
                     ->  Sort  (cost=138.66..143.66 rows=2000 width=8)
                           Sort Key: s.b
                           ->  Seq Scan on s  (cost=0.00..29.00 rows=2000 width=8)
                     ->  Sort  (cost=1717.77..1767.77 rows=20000 width=8)
                           Sort Key: r.b
                           ->  Seq Scan on r  (cost=0.00..289.00 rows=20000 width=8)
(16 rows)

u11924480=> set enable_mergejoin=0;
SET
u11924480=> set enable_nestloop=1;
SET
u11924480=> EXPLAIN SELECT a,b,c FROM r NATURAL JOIN s NATURAL JOIN t;
                                 QUERY PLAN
-----------------------------------------------------------------------------
 Nested Loop  (cost=0.00..141915612.00 rows=1983849 width=12)
   Join Filter: ((r.a = t.a) AND (s.c = t.c))
   ->  Nested Loop  (cost=0.00..600323.00 rows=403757 width=12)
         Join Filter: (r.b = s.b)
         ->  Seq Scan on r  (cost=0.00..289.00 rows=20000 width=8)
         ->  Materialize  (cost=0.00..39.00 rows=2000 width=8)
               ->  Seq Scan on s  (cost=0.00..29.00 rows=2000 width=8)
   ->  Materialize  (cost=0.00..389.00 rows=20000 width=8)
         ->  Seq Scan on t  (cost=0.00..289.00 rows=20000 width=8)
```

Based on this fact we can assume that it can affect computational and efficiency of you DBMS which could lead to slower processing. One more detail noticed is that query planer does not always pick by default the best solution.

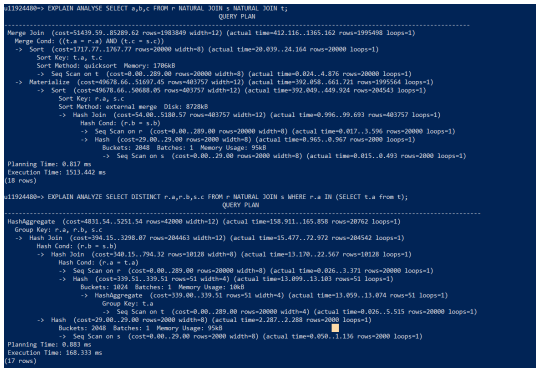
4.3 Add indexes to the DB that you believe may be useful for the query

Well the tables used in my case are not the best suitable for indexing but I tried adding btree indexes and retry the query plan mentioned above to see any progress. However, I did not see any major improvements in the performance of the query, although it did perform a bit faster. However, this might be the case that the indexes that I used do not provide a lot of help for the DBMS.

4.4 Reformulating the query

In order to try and optimise as much as possible and also to reduce execution time as much as possible i tried to use the reformulated query also described in the exercise description as we can see below:

```
EXPLAIN ANALYZE
SELECT DISTINCT r.a,r.b,s.c
FROM r NATURAL JOIN s WHERE r.a IN (SELECT t.a from t);
```



As it can be seen from the picture we have achieved to reduce the execution time of the query for Execution Time: 168.333 ms which seems to be the best result so far and really beyond my expectations.

4.5 Investigating the Joins

As it was also in the upper part on comparing performance of joints i tried enabling and disabling different joins in order to compare the performances, as it can be seen

that this approach does not give us any different results so change is not needed at this point.

4.6 Changing the relation joins

As required in this past i have made the changes presenting in the query below:

```
EXPLAIN ANALYZE
SELECT DISTINCT r.a,r.b,s.c
FROM r NATURAL JOIN s WHERE r.a IN (SELECT t.a from t);
```

After executing this query i obtained a very big values such as 2255.502 in terms of execution time compared to the previous query in d.

5 Exercise 4 : Optimizing Queries

5.1 a

The query provided to us:

```
SELECT count(*) FROM users
WHERE substring(displayname from '%#"[pbl][aeo]{2,}#"%' for '#') IS NOT NULL;
```

Execution time: 106.007 ms

Improved optimised query:

```
SELECT count (id) FROM users
WHERE substring (displayname from '%#"[pbl][aeo]{2,}#"%' for '#' )
IS NOT NULL ;
```

Execution time: 98.431 ms

Important to note that in these cases of comparisons its very important for better understanding to use EXPLAIN ANALYZE before the query. We can see from the example above that if we project just the ID instead of all the values query executes faster. This is a very important if we are working with indexing and in this case if we index the id for table users.

5.2 b

The query provided to us:

```
SELECT b.name, b.class FROM badges b
WHERE b.userid IN (SELECT u.id from users u WHERE u.reputation < b.class);
```

Execution time: Very long!!!

Improved optimised query:

```
EXPLAIN ANALYZE SELECT b.name, b.class FROM badges b
WHERE EXISTS ( SELECT u.id from users u
WHERE b.userid=u.id AND u.reputation < b.class);
```

Execution time: 37.615 ms

From the command IN used in the query we search all the specific table in order to get all the typles that satisfy the condition since we have the id column of users as a primary key we know that this query will always find one tuple. For this purpose its possible to get use of the EXISTS command which will stop the user and machines once it finds the first tuple and once it finds it it satisfies the condition and stops. It also makes a huge difference under 100ms threshhold.

5.3 c

The query provided to us:

```
SELECT distinct b.userid FROM badges b
WHERE (SELECT count(distinct b2.name) FROM badges b2
WHERE b2.name ILIKE 'so%' AND b2.userid=b.userid)
=
(SELECT count(distinct b2.name)
FROM badges b2 WHERE b2.name ILIKE 'so%');
```

Execution time: Very long!!!

Improved optimised query:

```
EXPLAIN ANALYZE SELECT b.name, b.class FROM badges b
WHERE EXISTS ( SELECT u.id from users u
WHERE b.userid=u.id AND u.reputation < b.class);
```

Execution time: 141.524 ms

In the first query we had a very long time to obtain results, as after the optimisation the execution time as we can see reduced leading us to conclusion on a more efficient query.

5.4 d

The query provided to us:

```
SELECT distinct p.* FROM users u, posthistory h, posts p, badges b
WHERE u.id = h.userid
```

```
AND h.userid = u.id AND h.userdisplayname = u.displayname
AND p.id = h.postid
AND b.userid = u.id
AND (b.name SIMILAR TO 'Necromancer' OR b.name SIMILAR TO 'Scholar')
AND (SELECT AVG(u2.upvotes)
FROM users u2
WHERE u2.creationdate > p.creationdate)
>=
(SELECT COUNT(*) FROM votes v
WHERE v.votetypeid = 15 AND v.postid = h.postid AND h.ContentLicense LIKE '%4.0%');
```

Execution time: 5494.397 ms

Improved optimised query:

Execution time: