Exercise Sheet 1, 2023

6.0 VU Advanced Database Systems

Teodor Chakarov (12141198)

24. März 2023

Exercise 1 (Disk Access) *[3 Points]* In the first Exercise we were given 2 disks (SSD and Magnetic). We have to calculate the I/O time (assuming the files are not in the main memory). The following data has being given:

1. Disk A (Magnetic)
   - Block size: 4 KB
   - Rotational speed: 10 000 rpm
   - Average seek time: 5ms
   - Transfer rate: 205 MB/s
   - Track-to-track seek time: 2ms
   - Average track size 500 KB
2. Disk B (SSD)
   - Block size: 1 MB
   - Transfer rate: 800 MB/s

We assume that DBMS is configured to store data in an unspanned organization in 500 KB blocks. If records are not allowed to cross block boundaries, the organisation is called unspanned, compared to spanned mapping where records can span more than one block. We consider the files **course and department**, referred to as c and d, respectively. The number of records n and record size R for each file is given as follows:
• nc = 501 Rc = 10 000 bytes = 10KB
• nd = 15 Rd = 25 500 bytes = 25,5KB
First in order to calculate the I/O time we have to explain the main variables.

- t_s: Seek time. Time to position head on the track.
- t_r: Rotational delay. Waiting time for desired block to rotate under postilion head. We can assume that delay to be half the time of a rotation.

- t_{tr}: Transfer time. This is the time to read or write data from/onto the block. This time depends on the rotational speed, track size, and block size. The formula is: *number of physical blocks* * $\frac{sizeofblocks}{transferrate}$.
- t_{t2t}: Track to track seek time. Time needed for arm to move to next track. Formula (for sequential reading approach): *(Number of tracks needed - 1) . Track to track seek time*
- t_a: Access time is the sum of all the above times added together as follows: $t_a = t_s + t_r + t_{tr} + t_{t2t}$

For calculation of records per block we have the following formula: $\frac{BlockSizeDBMS}{RecordSize}$
Number of records per DBMS Block:

- course: $\frac{500KB}{10KB}$ = 50KB records per block
- department: $\frac{500KB}{25{,}5KB}$ = 19,61KB records per block

For **fitting the whole records** we have to use the following formula:
$\frac{Numberofrecords}{Recordsperblock}$

- course: $\frac{501}{50KB}$ = 50,02 blocks for all the records
- department: $\frac{15}{25{,}5KB}$ = 0,59 blocks for all the records

In order to calculate the access times for both disks we have to know that they are different technology.

(a) We want to calculate basic access times of the Seq Scan, see Figure 1, which basically scans the whole table/relation as stored on disk.
Disk A and course:
- Records per page - 50KB
- Pages needed - $\frac{501records}{50recordsperpage}$ = 10,02 => 11 pages needed
- Number of physical blocks needed: $\frac{500KBpagesize*11pagesneeded}{4KB/blocksize}$ = 1375 physical blocks.
- t_s - 5ms
- t_r - 0,5 * $\frac{60{,}000millisecondsinmin}{10{,}000rpm}$ = 3ms
- t_{tr} - $\frac{4KB}{205000kb/ms}$ * 1375 = 0.02ms
- Tracks needed for server - $\frac{Numberofrecords*sizeofrecord}{averagetracksize}$ -> $\frac{501*10}{500}$ = 11 tracks
- t_{t2t} - 11 tracks - 1(because sequential) = 10ms
- **Access time: 5 + 3 + 0,02*1375 + 2ms*10 = 55,5ms**

Disk A and department:
- Records per page - 19,61KB
- Pages needed - $\frac{15records}{19{,}61MBrecordsperpage}$ = 0,05 1 page needed
- Number of physical blocks needed: $\frac{500KBpagesize*1pageneeded}{4KB/blocksize}$ = 125 physical blocks.
- t_s - 5ms

- t_r - 0,5 * $\frac{60{,}000 milliseconds_i n_m in}{10{,}000 rpm}$ = 3ms
- t_{tr} - $\frac{4KB}{205ms/s * 10000kb/ms}$ * 1375 number of physical blocks = 0.02ms
- Tracks needed for server - $\frac{Numberofrecords * sizeofrecord}{averagetracksize}$ -> $\frac{15*25{,}5KB}{500}$ = 0,76 1 track
- Average t_{t2t} - 2ms * (1 track - 1(because sequential)) = 0ms
- **Access time: 5 + 3 + 0,02*125 + 0 = 10,5ms**

Disk B
- No seek time
- No rotation delay
- Access time of one block $a_t = \frac{Dataread/write}{transferrate} = \frac{1000}{800MB/s}$ = 1,25ms
- **course sequential scan = total tracks needed * 1,25 = 11 * 1,25 = 13,75**
- **department sequential scan = total tracks needed * 1,25 = 1 * 1,25 = 1,25**

13,75 + 1,25 = 15ms total for SSD

Comparison
For comparison Disk A and disk B we can see that SSD technology has significant less access time than the Magnetic Disk.

(b) A database index is a data structure that improves the speed of data retrieval operations on a database table at the cost of additional writes and storage space to maintain the index data structure. The query planner has now using an index scan instead of a sequential scan to read the records from the file. An Index Scan looks up a value in an index and reads the respective entry to which the index points. department.id is looked up in an index for course.by. Entries are only read if the department.id occurs in the index.
Size of the disk 2300 KB. There are 45 rows in the result of the query
Disk A We will calculate the Index which is department:
- Blocks needed for index: Size of index/ Block size -> $\frac{2300}{500KB}$ = 4,6 => 5
- Physical blocks needed for Disk A: 5 * 125(department physical blocks) = 625
- Number of track changes for index: $\frac{PhysicalBlocksForIndex * BlockSize}{AverageTrackSize}$ - 1 = $\frac{625*4}{500}$ - 1 = 5 - 1 = 4

Now calculate access time for index and course:
- We need to calculate index and course (department not needed since we have the index of it):

Index: $t_a = t_s + t_r + t_{tr} + t_{t2t}$ = 5 + 3 + 625 (physical blocks needed for index) 4(block size MB) /205(transfer time) + 4(track changes 2(track changes avg time) = 28.2ms
Course: t_a = 5 + 3 + 1375 * 4/205 + 10*2 = 54,83 ms
Access time for 45 rows: Number of records * t_a = 45 * (5+3+4/205)=360.88ms
Total time: 360.88+54,83+28.2=443,91 ms

Disk B - Blocks needed for index: 5 * 500 (Average Track Size) / 1000 (Block size KB)= 3
- Blocks needed for Course: 501 * 10 /1000=5.01 6

- Access time for index: 1.25*3= 3.75ms
- Access time for course: 1.25*6= 7.5ms
- Total for 45rows: 45*1.25+3.75+7.5= 67.5ms

Exercise 2 (Selectivity) *[3 Points]*

(a) Estimate the selectivity for the following two predicates.

- performances < 3
  We have equally spread data based on 7 groups. Lower value is 0 and highest 871. The following ranges are given: 1, 4, 9, 15, 35, 278.
  We have 22 000 values in total which is 3143 values in each range. We also assume that the values are equally spread in each range. We have values from the first bucket 0 to 1 and values from the second from 1 to 2 which are inside the selected values. Which means 3143 + $\frac{2}{3}$*3143 / 22000 = 0,24 Which means high selectivity.
- performances > 100
  Now we will have 5 full buckets (0-1, 1-4, 4-9, 9-15, 15-35) and (35-100). 5*3143 = 15715 values and (100-35)/(278-35)*3143 = 840,45.
  And the total selectivity is (15715+840,45) / 22000 = 0,75 and now 1- 0,75 = 0,25 to reverse the order. Which is big selectivity.

(b) Calculating selectivity for single value

- performances = 356
  With equal approximation we get: 1/523 = 0,0019. That means the selectivity is really high since its one value.
- performances != 1
  Which not including 1 value we have 522/523 = 0,998. Low selectivity.

(c) Estimate the selectivity with second table

- opera $\bowtie_{id=opera}$performance
  Because we have only one value which can match the join operation (because of the foreign key), we have the following:
  1/102 000 = 0,0001
- $\pi opera$(performance)
  Here we get as selectivity 1, because its just projection on the column and all values are selected => low selectivity.

(d) Ordering of joins and selection operations
We have to calculate the selectivity for each of the filter methods.

- p.city = 1/50 000 = 0,00002
- p.number = 1 => its first bucket so we have 255 000 values in each bucket. 255 000 / 2 = 12750 number of ones. 12750/102000 = 0,125 selectivity for p.number

In total for the **(p.city = 'Vienna' OR p.number = 1)** we have 0,12502 as selectivity.

- d.operas < 3
  We have 5 buckets / 500 = 100. Then we got 100 + 1/2*100 = 150 / 500 = 0,3 as selectivity.
- o.actors > 20
  We have 6 buckets for 22000 values we get 3667 for each bucket. We have 4*3667 = 14668 + (20-15)/(35-15)*3667 = 14668 + 916,75 / 22 000 = 0,70 and when we change the direction 1 - 0,7 = 0,3 as selectivity (d.operas < 3 or p.number = 1)

In general we can go from the larger selectivity to the less selectivity. We can apply the filters like follows:
(p.city = 'Vienna' OR p.number = 1)
Then we chose the next one since both of them are 0,3 selectivity.

(e) Optimizing the query
We can do as follows:

a) Filter (p.city = 'Vienna' OR p.number = 1)

b) Filter opera < 3

c) Join performance with opera

d) Filter Actors > 20

e) Join actors with the rest table

Exercise 3 (The Query Planner and You) *[4 Points]*

(a)

```
Unique  (cost=102161.54..102449.34 rows=57560 width=32) (actual time=7655.041..8048.067 rows=9934 loops=1)
  →  Sort  (cost=102161.54..102305.44 rows=57560 width=32) (actual time=7655.039..7875.088 rows=122460 loops=1)
       Sort Key: (ROW(r.a, r.b, r.c, s.d))
       Sort Method: external merge  Disk: 5648kB
       →  Gather  (cost=88752.06..97610.62 rows=57560 width=32) (actual time=4442.698..5769.506 rows=122460 loops=1)
            Workers Planned: 1
            Workers Launched: 0
            →  Merge Join  (cost=87752.06..90854.62 rows=33859 width=32) (actual time=4441.598..5710.049 rows=122460 loops=1)
                 Merge Cond: ((r.c = s.c) AND (r.b = s.b) AND (r.a = t.a))
                 →  Sort  (cost=37388.47..38123.76 rows=294118 width=12) (actual time=2038.933..2623.083 rows=497013 loops=1)
                      Sort Key: r.c, r.b, r.a
                      Sort Method: external merge  Disk: 10800kB
                      →  Parallel Seq Scan on r  (cost=0.00..5644.18 rows=294118 width=12) (actual time=0.022..389.959 rows=500000 loops=1)
                 →  Sort  (cost=50363.58..50401.33 rows=15100 width=28) (actual time=2402.569..2503.826 rows=123139 loops=1)
                      Sort Key: s.c, s.b, t.a
                      Sort Method: quicksort  Memory: 3203kB
                      →  Merge Join  (cost=46947.93..49315.46 rows=15100 width=28) (actual time=1814.200..2296.615 rows=31165 loops=1)
                           Merge Cond: ((u.b = s.b) AND (t.c = s.c) AND (t.d = s.d))
                           →  Sort  (cost=22836.61..23224.93 rows=155328 width=24) (actual time=1043.002..1245.974 rows=155655 loops=1)
                                Sort Key: u.b, t.c, t.d
                                Sort Method: external merge  Disk: 5184kB
                                →  Merge Join  (cost=3475.54..6254.99 rows=155328 width=24) (actual time=192.916..356.796 rows=155655 loops=1)
                                     Merge Cond: ((t.a = u.a) AND (t.d = u.d))
                                     →  Sort  (cost=1737.77..1787.77 rows=20000 width=12) (actual time=85.909..89.235 rows=7936 loops=1)
                                          Sort Key: t.a, t.d
                                          Sort Method: quicksort  Memory: 1706kB
                                          →  Seq Scan on t  (cost=0.00..309.00 rows=20000 width=12) (actual time=0.045..21.979 rows=20000 loops=1)
                                     →  Sort  (cost=1737.77..1787.77 rows=20000 width=12) (actual time=106.993..178.218 rows=155646 loops=1)
                                          Sort Key: u.a, u.d
                                          Sort Method: quicksort  Memory: 1706kB
                                          →  Seq Scan on u  (cost=0.00..309.00 rows=20000 width=12) (actual time=0.049..58.134 rows=20000 loops=1)
                           →  Materialize  (cost=24111.14..25111.14 rows=200000 width=12) (actual time=771.106..877.846 rows=67191 loops=1)
                                →  Sort  (cost=24111.14..24611.14 rows=200000 width=12) (actual time=771.046..836.204 rows=40037 loops=1)
                                     Sort Key: s.b, s.c, s.d
                                     Sort Method: external merge  Disk: 4320kB
                                     →  Seq Scan on s  (cost=0.00..3082.00 rows=200000 width=12) (actual time=0.118..166.980 rows=200000 loops=1)
Planning Time: 2.623 ms
Execution Time: 8057.746 ms
```

The default join strategy is Merge Join with running time 8057.45ms.

(b) Hash join:

```
 Unique  (cost=674943.84..675231.64 rows=57560 width=32) (actual time=4892.616..5558.571 rows=9934 loops=1)
   →  Sort  (cost=674943.84..675087.74 rows=57560 width=32) (actual time=4892.614..5332.893 rows=122460 loops=1)
         Sort Key: (ROW(r.a, r.b, r.c, s.d))
         Sort Method: external merge  Disk: 5648kB
         →  Gather  (cost=469200.84..670392.92 rows=57560 width=32) (actual time=1622.864..2730.570 rows=122460 loops=1)
               Workers Planned: 1
               Workers Launched: 0
               →  Hash Join  (cost=468200.84..663636.92 rows=33859 width=32) (actual time=1622.204..2710.382 rows=122460 loops=1)
                     Hash Cond: ((r.b = s.b) AND (r.c = s.c) AND (r.a = t.a))
                     →  Parallel Seq Scan on r  (cost=0.00..5644.18 rows=294118 width=12) (actual time=0.035..418.919 rows=500000 loops=1)
                     →  Hash  (cost=467936.59..467936.59 rows=15100 width=28) (actual time=1621.948..1621.954 rows=31165 loops=1)
                           Buckets: 32768 (originally 16384)  Batches: 1 (originally 1)  Memory Usage: 2083kB
                           →  Hash Join  (cost=135974.59..467936.59 rows=15100 width=28) (actual time=1382.443..1607.695 rows=31165 loops=1)
                                 Hash Cond: ((u.b = s.b) AND (u.a = t.a) AND (u.d = s.d))
                                 →  Seq Scan on u  (cost=0.00..309.00 rows=20000 width=12) (actual time=0.026..6.223 rows=20000 loops=1)
                                 →  Hash  (cost=126882.85..126882.85 rows=389185 width=24) (actual time=1382.068..1382.071 rows=396666 loops=1)
                                       Buckets: 65536  Batches: 8  Memory Usage: 3225kB
                                       →  Hash Join  (cost=7059.00..126882.85 rows=389185 width=24) (actual time=378.785..973.090 rows=396666 loops=1)
                                             Hash Cond: ((t.c = s.c) AND (t.d = s.d))
                                             →  Seq Scan on t  (cost=0.00..309.00 rows=20000 width=12) (actual time=0.027..7.088 rows=20000 loops=1)
                                             →  Hash  (cost=3082.00..3082.00 rows=200000 width=12) (actual time=378.368..378.369 rows=200000 loops=1)
                                                   Buckets: 131072  Batches: 4  Memory Usage: 3161kB
                                                   →  Seq Scan on s  (cost=0.00..3082.00 rows=200000 width=12) (actual time=0.039..117.858 rows=200000 loop
s=1)
 Planning Time: 1.439 ms
 Execution Time: 5561.916 ms
(25 rows)
```

As running time we have 5561ms
Merge Join:

```
Unique  (cost=102161.54..102449.34 rows=57560 width=32) (actual time=7052.811..7620.713 rows=9934 loops=1)
  →  Sort  (cost=102161.54..102305.44 rows=57560 width=32) (actual time=7052.809..7427.379 rows=122460 loops=1)
        Sort Key: (ROW(r.a, r.b, r.c, s.d))
        Sort Method: external merge  Disk: 5648kB
        →  Gather  (cost=88752.06..97610.62 rows=57560 width=32) (actual time=4271.572..5344.495 rows=122460 loops=1)
              Workers Planned: 1
              Workers Launched: 0
              →  Merge Join  (cost=87752.06..90854.62 rows=33859 width=32) (actual time=4270.898..5288.636 rows=122460 loops=1)
                    Merge Cond: ((r.c = s.c) AND (r.b = s.b) AND (r.a = t.a))
                    →  Sort  (cost=37388.47..38123.76 rows=294118 width=12) (actual time=2042.801..2556.157 rows=497013 loops=1)
                          Sort Key: r.c, r.b, r.a
                          Sort Method: external merge  Disk: 10800kB
                          →  Parallel Seq Scan on r  (cost=0.00..5644.18 rows=294118 width=12) (actual time=0.028..347.928 rows=500000 loops=1)
                    →  Sort  (cost=50363.58..50401.33 rows=15100 width=28) (actual time=2228.073..2275.105 rows=123139 loops=1)
                          Sort Key: s.c, s.b, t.a
                          Sort Method: quicksort  Memory: 3203kB
                          →  Merge Join  (cost=46947.93..49315.46 rows=15100 width=28) (actual time=1707.888..2138.132 rows=31165 loops=1)
                                Merge Cond: ((u.b = s.b) AND (t.c = s.c) AND (t.d = s.d))
                                →  Sort  (cost=22836.61..23224.93 rows=155328 width=24) (actual time=949.550..1019.681 rows=155655 loops=1)
                                      Sort Key: u.b, t.c, t.d
                                      Sort Method: external merge  Disk: 5184kB
                                      →  Merge Join  (cost=3475.54..6254.99 rows=155328 width=24) (actual time=108.018..343.096 rows=155655 loops=1)
                                            Merge Cond: ((t.a = u.a) AND (t.d = u.d))
                                            →  Sort  (cost=1737.77..1787.77 rows=20000 width=12) (actual time=50.506..52.134 rows=7936 loops=1)
                                                  Sort Key: t.a, t.d
                                                  Sort Method: quicksort  Memory: 1706kB
                                                  →  Seq Scan on t  (cost=0.00..309.00 rows=20000 width=12) (actual time=0.049..16.775 rows=20000 loops=1)
                                            →  Sort  (cost=1737.77..1787.77 rows=20000 width=12) (actual time=57.499..135.236 rows=155646 loops=1)
                                                  Sort Key: u.a, u.d
                                                  Sort Method: quicksort  Memory: 1706kB
                                                  →  Seq Scan on u  (cost=0.00..309.00 rows=20000 width=12) (actual time=0.049..45.566 rows=20000 loops=1)
                                →  Materialize  (cost=24111.14..25111.14 rows=200000 width=12) (actual time=758.307..845.088 rows=67191 loops=1)
                                      →  Sort  (cost=24111.14..24611.14 rows=200000 width=12) (actual time=758.298..814.002 rows=40037 loops=1)
                                            Sort Key: s.b, s.c, s.d
                                            Sort Method: external merge  Disk: 4320kB
                                            →  Seq Scan on s  (cost=0.00..3082.00 rows=200000 width=12) (actual time=0.041..135.743 rows=200000 loops=1)
Planning Time: 2.049 ms
Execution Time: 7631.315 ms
```

With running time 7631ms
The nested loop was running too much and I cancelled the work of it.

(c) After I add the Indexes as the given picture:

```
u12141198=> CREATE INDEX r_idx on r (a, b, c);
CREATE INDEX
u12141198=> CREATE INDEX s_idx on s (d, b, c);
CREATE INDEX
u12141198=> CREATE INDEX t_idx on t (a, d, c);
CREATE INDEX
u12141198=> CREATE INDEX u_idx on u (a, b, d);
CREATE INDEX
u12141198=> \di+
                                    List of relations
 Schema |         Name          | Type  |   Owner    |      Table       |    Size    | D
--------+-----------------------+-------+------------+------------------+------------+--
 public | basiertauf_pkey       | index | u11809919 | basiertauf       | 8192 bytes |
 public | bestehtaus_pkey       | index | u11809919 | bestehtaus       | 8192 bytes |
 public | gutscheincode_pkey    | index | u11809919 | gutscheincode    | 8192 bytes |
 public | haendler_pkey         | index | u11809919 | haendler         | 8192 bytes |
 public | hersteller_pkey       | index | u11809919 | hersteller       | 8192 bytes |
 public | kaffee_pkey           | index | u11809919 | kaffee           | 8192 bytes |
 public | kaffeekapsel_pkey     | index | u11809919 | kaffeekapsel     | 8192 bytes |
 public | kaffeekapseltyp_pkey  | index | u11809919 | kaffeekapseltyp  | 8192 bytes |
 public | kaffeemaschine_pkey   | index | u11809919 | kaffeemaschine   | 8192 bytes |
 public | kannzubereiten_pkey   | index | u11809919 | kannzubereiten   | 8192 bytes |
 public | kompatibelmit_pkey    | index | u11809919 | kompatibelmit    | 8192 bytes |
 public | lizenz_pkey           | index | u11809919 | lizenz           | 8192 bytes |
 public | mengenrabatt_pkey     | index | u11809919 | mengenrabatt     | 8192 bytes |
 public | produkt_pkey          | index | u11809919 | produkt          | 8192 bytes |
 public | r_idx                 | index | u12141198 | r                | 15 MB      |
 public | review_pkey           | index | u11809919 | review           | 8192 bytes |
 public | s_idx                 | index | u12141198 | s                | 6184 kB    |
 public | t_idx                 | index | u12141198 | t                | 632 kB     |
 public | u_idx                 | index | u12141198 | u                | 632 kB     |
 public | unterstuetzt_pkey     | index | u11809919 | unterstuetzt     | 8192 bytes |
 public | verkauft_pkey         | index | u11809919 | verkauft         | 8192 bytes |
(21 rows)
```

The improvement of the times wasnt that great. I got the following improvement just for the merge join.

```
 Unique  (cost=65451.10..65738.90 rows=57560 width=32) (actual time=4545.888..5018.858 rows=9934 loops=1)
   ->  Sort  (cost=65451.10..65595.00 rows=57560 width=32) (actual time=4545.880..4835.027 rows=122460 loops=1)
         Sort Key: (ROW(r.a, r.b, r.c, s.d))
         Sort Method: external merge  Disk: 5648kB
         ->  Merge Join  (cost=30832.86..60900.19 rows=57560 width=32) (actual time=1988.969..3202.889 rows=122460 loops=1)
               Merge Cond: ((r.a = t.a) AND (r.b = s.b) AND (r.c = s.c))
               ->  Index Only Scan using r_idx on r  (cost=0.42..26020.89 rows=500000 width=12) (actual time=0.088..885.069 rows=199328 loops=1)
                     Heap Fetches: 199328
               ->  Sort  (cost=30832.43..30870.18 rows=15100 width=28) (actual time=1988.698..2023.368 rows=123138 loops=1)
                     Sort Key: t.a, s.b, s.c
                     Sort Method: quicksort  Memory: 3203kB
                     ->  Merge Join  (cost=21508.40..29784.32 rows=15100 width=28) (actual time=1033.701..1838.219 rows=31165 loops=1)
                           Merge Cond: ((s.d = t.d) AND (s.b = u.b) AND (s.c = t.c))
                           ->  Index Only Scan using s_idx on s  (cost=0.42..10420.27 rows=200000 width=12) (actual time=0.014..333.910 rows=39709 loops=1)
                                 Heap Fetches: 39709
                           ->  Materialize  (cost=21507.98..22284.62 rows=155328 width=24) (actual time=1033.668..1223.339 rows=158442 loops=1)
                                 ->  Sort  (cost=21507.98..21896.30 rows=155328 width=24) (actual time=1033.663..1151.412 rows=155655 loops=1)
                                       Sort Key: t.d, u.b, t.c
                                       Sort Method: external merge  Disk: 5184kB
                                       ->  Merge Join  (cost=1738.06..4926.36 rows=155328 width=24) (actual time=57.653..399.837 rows=155655 loops=1)
                                             Merge Cond: ((t.a = u.a) AND (t.d = u.d))
                                             ->  Index Only Scan using t_idx on t  (cost=0.29..1052.25 rows=20000 width=12) (actual time=0.039..32.730 rows=
7936 loops=1)
                                                   Heap Fetches: 7936
                                             ->  Sort  (cost=1737.77..1787.77 rows=20000 width=12) (actual time=57.604..132.207 rows=155646 loops=1)
                                                   Sort Key: u.a, u.d
                                                   Sort Method: quicksort  Memory: 1706kB
                                                   ->  Seq Scan on u  (cost=0.00..309.00 rows=20000 width=12) (actual time=0.024..5.994 rows=20000 loops=1)
 Planning Time: 3.789 ms
 Execution Time: 5043.303 ms
(29 rows)
```

The improvement was from 7631 to 5043ms. The hash join without index was 5561 and with was 6235ms.

(d) The new query's execution time is 4648ms which is way faster than the first query being run

```
Unique  (cost=38286.28..39062.92 rows=102000 width=32) (actual time=4552.473..4623.149 rows=9934 loops=1)
  ->  Sort  (cost=38286.28..38674.60 rows=155328 width=32) (actual time=4552.470..4556.732 rows=27742 loops=1)
        Sort Key: (ROW(u.a, u.b, t.c, u.d))
        Sort Method: quicksort  Memory: 2936kB
        ->  Merge Join  (cost=20454.93..21176.15 rows=155328 width=32) (actual time=3957.087..4122.942 rows=27742 loops=1)
              Merge Cond: ((t.a = u.a) AND (t.d = u.d) AND (t.c = r.c))
              ->  Index Only Scan using t_idx on t  (cost=0.29..1052.25 rows=20000 width=12) (actual time=0.053..36.964 rows=7936 loops=1)
                    Heap Fetches: 7936
              ->  Sort  (cost=20454.64..20504.64 rows=20000 width=28) (actual time=3956.972..4010.721 rows=180693 loops=1)
                    Sort Key: u.a, u.d, r.c
                    Sort Method: external sort  Disk: 7352kB
                    ->  Hash Join  (cost=17910.00..19025.87 rows=20000 width=28) (actual time=2365.003..3374.873 rows=178514 loops=1)
                          Hash Cond: ((u.a = r.a) AND (u.b = r.b) AND (s.c = r.c))
                          ->  Hash Join  (cost=5082.00..5935.57 rows=20000 width=24) (actual time=775.130..1156.360 rows=357495 loops=1)
                                Hash Cond: ((u.d = s.d) AND (u.b = s.b))
                                ->  Seq Scan on u  (cost=0.00..309.00 rows=20000 width=12) (actual time=0.020..4.792 rows=20000 loops=1)
                                ->  Hash  (cost=4782.00..4782.00 rows=20000 width=12) (actual time=774.927..774.929 rows=181628 loops=1)
                                      Buckets: 131072 (originally 32768)  Batches: 4 (originally 1)  Memory Usage: 3073kB
                                      ->  HashAggregate  (cost=4582.00..4782.00 rows=20000 width=12) (actual time=394.521..645.899 rows=181628 loops=1)
                                            Group Key: s.b, s.c, s.d
                                            ->  Seq Scan on s  (cost=0.00..3082.00 rows=200000 width=12) (actual time=0.021..169.697 rows=200000 loops=1)
                          ->  Hash  (cost=11953.00..11953.00 rows=50000 width=12) (actual time=1589.652..1589.654 rows=128400 loops=1)
                                Buckets: 131072 (originally 65536)  Batches: 2 (originally 1)  Memory Usage: 3779kB
                                ->  HashAggregate  (cost=11453.00..11953.00 rows=50000 width=12) (actual time=1275.908..1447.180 rows=128400 loops=1)
                                      Group Key: r.a, r.b, r.c
                                      ->  Seq Scan on r  (cost=0.00..7703.00 rows=500000 width=12) (actual time=0.037..409.726 rows=500000 loops=1)
Planning Time: 3.426 ms
Execution Time: 4648.783 ms
(28 rows)
```

The new query is joining different tables but the filtering has high selectivity which makes less tuples to be computed later on the joins. The query is following:
SELECT distinct(u.a, u.b, t.c, u.d) FROM u NATURAL JOIN t
WHERE EXISTS
(SELECT DISTINCT(r.a,r.b,r.c) FROM r WHERE t.a=r.a AND u.b=r.b AND t.c=r.c) AND EXISTS
(SELECT DISTINCT(s.b,s.c,s.d) FROM s WHERE u.b=s.b AND t.c=s.c AND s.d=u.d);

(e) The planner is using the Hash aggregate on grouped keys which is also reduction on the tuples on the table as well the sort function. Based on the planner it is already hash join.

(f) The reversed query where we change the places of the table should look like this: SELECT distinct(r.a, r.b, r.c, s.d) FROM r NATURAL JOIN s WHERE EXISTS (SELECT DISTINCT(t.a,t.d,t.c) FROM t WHERE r.a=t.a AND s.d=t.d AND r.c=t.c) AND EXISTS (SELECT DISTINCT(u.a,u.b,u.d) FROM u WHERE r.a=u.a AND r.b=u.b AND s.d=u.d);

```
Unique  (cost=52357.48..53223.91 rows=173286 width=32) (actual time=8190.483..8250.518 rows=9934 loops=1)
  ->  Sort  (cost=52357.48..52790.69 rows=173286 width=32) (actual time=8190.472..8220.572 rows=43559 loops=1)
        Sort Key: (ROW(r.a, r.b, r.c, s.d))
        Sort Method: external merge  Disk: 2008kB
        ->  Merge Join  (cost=31968.75..33131.67 rows=173286 width=32) (actual time=6330.317..7850.873 rows=43559 loops=1)
              Merge Cond: ((u.d = s.d) AND (r.b = s.b) AND (r.c = s.c) AND (r.a = t.a))
              ->  Sort  (cost=22486.42..22692.27 rows=82340 width=24) (actual time=5233.145..6194.549 rows=1533544 loops=1)
                    Sort Key: u.d, r.b, r.c, r.a
                    Sort Method: external merge  Disk: 51056kB
                    ->  Hash Join  (cost=509.00..15763.64 rows=82340 width=24) (actual time=30.616..1102.381 rows=1533544 loops=1)
                          Hash Cond: ((r.b = u.b) AND (r.a = u.a))
                          ->  Seq Scan on r  (cost=0.00..7703.00 rows=500000 width=12) (actual time=0.027..134.000 rows=500000 loops=1)
                          ->  Hash  (cost=479.00..479.00 rows=2000 width=12) (actual time=30.565..30.567 rows=7989 loops=1)
                                Buckets: 8192 (originally 2048)  Batches: 1 (originally 1)  Memory Usage: 408kB
                                ->  HashAggregate  (cost=459.00..479.00 rows=2000 width=12) (actual time=13.418..28.283 rows=7989 loops=1)
                                      Group Key: u.a, u.b, u.d
                                      ->  Seq Scan on u  (cost=0.00..309.00 rows=20000 width=12) (actual time=0.032..3.808 rows=20000 loops=1)
              ->  Sort  (cost=9482.27..9610.19 rows=51168 width=24) (actual time=1096.997..1131.954 rows=177833 loops=1)
                    Sort Key: s.d, s.b, s.c, t.a
                    Sort Method: external sort  Disk: 13656kB
                    ->  Hash Join  (cost=509.00..5480.18 rows=51168 width=24) (actual time=26.591..358.674 rows=366679 loops=1)
                          Hash Cond: ((s.c = t.c) AND (s.d = t.d))
                          ->  Seq Scan on s  (cost=0.00..3082.00 rows=200000 width=12) (actual time=0.042..51.298 rows=200000 loops=1)
                          ->  Hash  (cost=479.00..479.00 rows=2000 width=12) (actual time=26.488..26.489 rows=18503 loops=1)
                                Buckets: 32768 (originally 2048)  Batches: 1 (originally 1)  Memory Usage: 1052kB
                                ->  HashAggregate  (cost=459.00..479.00 rows=2000 width=12) (actual time=14.375..21.193 rows=18503 loops=1)
                                      Group Key: t.a, t.d, t.c
                                      ->  Seq Scan on t  (cost=0.00..309.00 rows=20000 width=12) (actual time=0.020..3.903 rows=20000 loops=1)
Planning Time: 2.339 ms
Execution Time: 8272.220 ms
30 rows)
```

In this case the run time was slower than the previous with nearly a half of it.

Exercise 4 (Query Optimization) *[5 Points]*

(a) Optimize with Index

```
u12141198=⇒ CREATE INDEX users_displayname_partial_idx ON users (displayname) WHERE displayname LIKE '%#moe{2,}#%' OR displayname LIKE '%#noe{2,}#%';
CREATE INDEX
u12141198=⇒ EXPLAIN ANALYZE SELECT count(*) FROM users WHERE displayname LIKE '%#moe{2,}#%' OR displayname LIKE '%#noe{2,}#%';
                                                                  QUERY PLAN
──────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────
 Aggregate  (cost=15.47..15.48 rows=1 width=8) (actual time=0.012..0.013 rows=1 loops=1)
   →  Bitmap Heap Scan on users  (cost=4.13..15.46 rows=3 width=0) (actual time=0.005..0.006 rows=0 loops=1)
         Recheck Cond: (((displayname)::text ~~ '%#moe{2,}#%'::text) OR ((displayname)::text ~~ '%#noe{2,}#%'::text))
         →  Bitmap Index Scan on users_displayname_partial_idx  (cost=0.00..4.13 rows=3 width=0) (actual time=0.002..0.003 rows=0 loops=1)
 Planning Time: 0.769 ms
 Execution Time: 0.116 ms
(6 rows)
```

In this case we are creating index on the table users with condition as from the query. The query remains the same.

(b) The optimized query is the following: SELECT b.name, b.class FROM badges b JOIN users u ON u.id = b.userid AND u.reputation < b.class;

```
 Hash Join  (cost=806.68..1387.78 rows=9085 width=14) (actual time=13.581..83.178 rows=2425 loops=1)
   Hash Cond: (b.userid = u.id)
   Join Filter: (u.reputation < b.class)
   Rows Removed by Join Filter: 22874
   →  Seq Scan on badges b  (cost=0.00..509.54 rows=27254 width=18) (actual time=0.042..46.873 rows=25299 loops=1)
   →  Hash  (cost=624.08..624.08 rows=14608 width=8) (actual time=13.443..13.444 rows=14394 loops=1)
         Buckets: 16384  Batches: 1  Memory Usage: 691kB
         →  Seq Scan on users u  (cost=0.00..624.08 rows=14608 width=8) (actual time=0.005..9.278 rows=14394 loops=1)
 Planning Time: 0.354 ms
 Execution Time: 83.419 ms
(10 rows)
```

(c) The running time is 12ms

```
u12141198=⇒ EXPLAIN ANALYZE SELECT b.userid FROM badges b WHERE b.name LIKE 'v%' GROUP BY b.userid HAVING count(DISTINCT b.name) = (SELECT count(DISTINCT name) FROM badges WHERE name LIKE 'v%');
                                                                  QUERY PLAN
──────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────
 GroupAggregate  (cost=1155.37..1155.39 rows=1 width=4) (actual time=12.354..12.356 rows=1 loops=1)
   Group Key: b.userid
   Filter: (count(DISTINCT b.name) = $0)
   InitPlan 1 (returns $0)
     →  Aggregate  (cost=577.68..577.69 rows=1 width=8) (actual time=5.422..5.423 rows=1 loops=1)
           →  Seq Scan on badges  (cost=0.00..577.67 rows=1 width=10) (actual time=3.572..5.409 rows=1 loops=1)
                 Filter: (name ~~ 'v%'::text)
                 Rows Removed by Filter: 25298
   →  Sort  (cost=577.68..577.69 rows=1 width=14) (actual time=6.819..6.820 rows=1 loops=1)
         Sort Key: b.userid
         Sort Method: quicksort  Memory: 25kB
         →  Seq Scan on badges b  (cost=0.00..577.67 rows=1 width=14) (actual time=4.321..6.809 rows=1 loops=1)
               Filter: (name ~~ 'v%'::text)
               Rows Removed by Filter: 25298
 Planning Time: 0.451 ms
 Execution Time: 12.440 ms
(16 rows)
```

We got rid of one nested query to optimize the performance.

(d)