Task 3 Report

Data Storage Paradigms, IV1351

Jonathan Nilsson Cullgert jonnc@kth.se

4 December 2023

1 Introduction

In this task, 4 different OLAP queries are meant to be created and tables which represent what these queries gather are also created. At least one of these queries have to be analyzed using EXPLAIN ANALYZE. Some changes are also made to the database and the data it contains to make it possible to create these queries.

The queries that are created are: the number of lessons given per month during a specified year, show how many students there are with 0, 1, 2, ... amount of siblings, listing the id and name of instructors teaching more than a specific number of lessons during the current month and all the ensembles being held during a specific week.

I did this assignment with Simon Malmqvist Dichy and Andrej Miladinovic.

2 Literature Study

I watched the lectures on SQL and read the tips and tricks document for task 3. I also spent some time reading the postgreSQL documentation so that I could understand the EXPLAIN ANALYZE command in postgreSQL. I learned how to make simple queries in SQL and that it is possible to have nested queries.

3 Method

This task was made using postgreSQL and pgadmin. The queries were able to be verified since all queries generated a table containing the data which was supposed to be found. The data in these tables were crosschecked with the data in the database and since both the table and the database provided the same result the queries worked as expected.

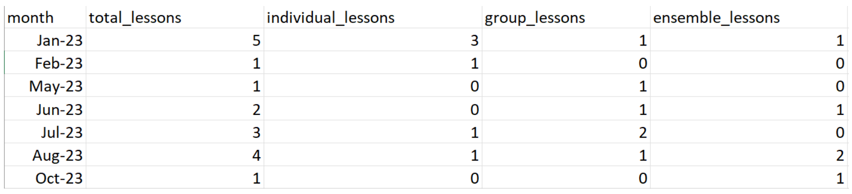
4 Result

The first query is supposed to extract the individual and group lessons, as well as the ensembles from every month during a specific year. In this query the given year is 2023 and it takes all those lessons and stores them in each column representing them. It will also take all lessons from the month, add them and show it in the total_lessons. Only months were lessons are held are shown in the table, that means months were no lessons are given are not in the table.

In the second query the data to show how many students have different amounts of siblings is generated. In the database every student is given a sibling id. All siblings share the same sibling id, which means that all sibling are sorted in sibling ”groups”. This query will check how many students have null in sibling id, these are students with 0 siblings. It will then go through all the sibling ids and check how many other students have the same sibling id and count them. As an example if two students are siblings they will be added to the table as 2 students with 1 sibling.

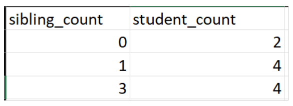
In the third query it is possible to find instructors that teach above a certain amount of lessons each month. This specific query will handle a given month, take a value as an input in this case it is 0. The query will then find all instructors that teach more lessons than that given input. In this example in figure 3 all instructors that taught more lessons than 0 during January 2023 will be found. The id for these instructors, their name as well as how many lessons they taught are shown in three columns.

The fourth and final query will, given a week, find all ensembles being taught that week. The data retrieved from this query will be stored in three columns. The day of the week, the genre and seat availability will be stored. The date of week is retrieved in SQL as 0 through 6, this is translated into actual weekdays with different cases. The case statement will also decide to print 0 if 0 seats are left, 1-2 if 1 or 2 seats are left and many seats if more than 2 seats are available.



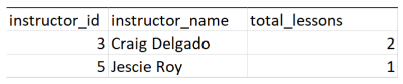
month	total_lessons	individual_lessons	group_lessons	ensemble_lessons
Jan-23	5	3	1	1
Feb-23	1	1	0	0
May-23	1	0	1	0
Jun-23	2	0	1	1
Jul-23	3	1	2	0
Aug-23	4	1	1	2
Oct-23	1	0	0	1

Figure 1: The table created from the query in task 1



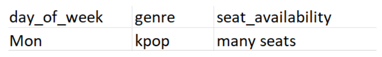
sibling_count	student_count
0	2
1	4
3	4

Figure 2: The table created from the query in task 2



instructor_id	instructor_name	total_lessons
3	Craig Delgado	2
5	Jescie Roy	1

Figure 3: The table created from the query in task 3



day_of_week	genre	seat_availability
Mon	kpop	many seats

Figure 4: The table created from the query in task 4

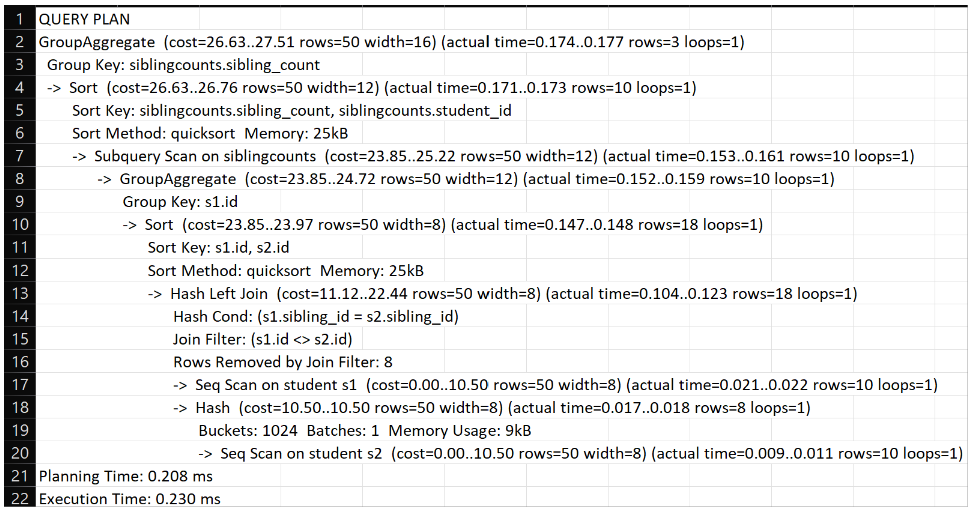
https://gits-15.sys.kth.se/jonnc/DatabaseIV1351/tree/main/task3

5 Discussion

This task required many changes to the database from the previous assignment since the queries that were requested in this task were not able to made, at least easily with the database that was created previously. Firstly the sibling entity was completely removed since the way it was implemented previously made it impossible to get the correct amount of siblings a student had. This created a lot of confusion when creating the query since we were never able to figure out how to check if a student had siblings without accidentally counting all the siblings too many times. We changed the database by removing the sibling entity as well as the many to many table between student and sibling. In the new database a sibling_id was created as an attribute in the student entity. This way all students will grouped in sibling groups, that is every student is given a sibling_id and if two or more students share the same sibling_id, they are siblings. This made the query to find how many students had a certain amount of siblings much easier.

Another big change that had to be made to the database was that the three different lesson entities had to be changed, all of them had to get the date attributed added to them. Ensemble also had to get an attribute added which tracked how many students that were currently signed up to an ensemble. The first change had to be made since it was not possible to the first query without having dates on the lessons. If there were no dates no data on was recorded on when a lesson was held, except for at which time. When this was added to the database the query was actually possible to create. The second change to ensemble was to have data on how many students are currently registered on an ensemble. This change was made to make it possible in the fourth query to see how many seats are available. The current amount of registered students minus the max amount of students gives how many spots are left and with that the seat availability.

To analyze the second query using EXPLAIN ANALYZE we start at the leaf of the tree, that is line 20. A sequential scan is done on s2, that is all the student siblings a student has. The data scanned is then hashed on line 18. Another sequential scan is done on line 17 but this time on s1. S1 and S2 are the same table just stored in two



```
1  QUERY PLAN
2  GroupAggregate  (cost=26.63..27.51 rows=50 width=16) (actual time=0.174..0.177 rows=3 loops=1)
3    Group Key: siblingcounts.sibling_count
4    ->  Sort  (cost=26.63..26.76 rows=50 width=12) (actual time=0.171..0.173 rows=10 loops=1)
5          Sort Key: siblingcounts.sibling_count, siblingcounts.student_id
6          Sort Method: quicksort  Memory: 25kB
7          ->  Subquery Scan on siblingcounts  (cost=23.85..25.22 rows=50 width=12) (actual time=0.153..0.161 rows=10 loops=1)
8                ->  GroupAggregate  (cost=23.85..24.72 rows=50 width=12) (actual time=0.152..0.159 rows=10 loops=1)
9                      Group Key: s1.id
10                     ->  Sort  (cost=23.85..23.97 rows=50 width=8) (actual time=0.147..0.148 rows=18 loops=1)
11                           Sort Key: s1.id, s2.id
12                           Sort Method: quicksort  Memory: 25kB
13                           ->  Hash Left Join  (cost=11.12..22.44 rows=50 width=8) (actual time=0.104..0.123 rows=18 loops=1)
14                                 Hash Cond: (s1.sibling_id = s2.sibling_id)
15                                 Join Filter: (s1.id <> s2.id)
16                                 Rows Removed by Join Filter: 8
17                                 ->  Seq Scan on student s1  (cost=0.00..10.50 rows=50 width=8) (actual time=0.021..0.022 rows=10 loops=1)
18                                 ->  Hash  (cost=10.50..10.50 rows=50 width=8) (actual time=0.017..0.018 rows=8 loops=1)
19                                       Buckets: 1024  Batches: 1  Memory Usage: 9kB
20                                       ->  Seq Scan on student s2  (cost=0.00..10.50 rows=50 width=8) (actual time=0.009..0.011 rows=10 loops=1)
21 Planning Time: 0.208 ms
22 Execution Time: 0.230 ms
```

Figure 5: The output from using EXPLAIN ANALYZE on query 2

different variables. The next thing is on line 13 where a hash left join is done on s1 and s2. On line 14 and 15 we can see the conditions for this join, s1 and s2 must have the same sibling id and they cannot be the same student. After this hash left join is done, sorting is done on line 10, on the student id in s1 and s2. This sort is done so that the group aggregate on line 8 can be done since that only handles sorted data. Line 8 does the grouping on the student id of s1. On line 7 a subquery scan is done on the subquery that is siblingcounts, it scans for the output of the subquery. The output of the subquery is then sort by the sort on line 4. Finally the root at line 2 groups the sibling_count from the subquery.

The DBMs spends the most time in the hash left join is made on the student table on itself. This can be seen by the fact that the cost is 11.12...22.44, which is equivalent to 0...11.32. This left join has two conditions, that the sibling id of both tables must be the same and that the student id cannot be the same. All rows in table s1 are then hashed into buckets. All the rows in table s2 that meet the conditions are also hashed into the same buckets. The buckets that have multiple entries are then those that meet the join. The hash is done on the sibling id on the rows in both tables. We can see 18 rows where operated on and this operation removed 9 of them, as seen on line 16. It makes sense that this took the longest since it is using a hash function on a very small data set, and hashing is not that efficient on small datasets.

6 Comments About the Course

I spent roughly 12 hours on this task.