SoundGood Music School Rapport

viktor björkén

November 2022

Introduction

In this rapport we are going to be working on how to create an OLAP (Online Analytical Processing), queries and views. This is to learn how we can analyse our work. We will use our SQL database with the INSERT data that we learn in the previous seminar to create our database. With this database we are going to construct queries for different purposes to analyze database.

Method

During this project we used PostgreSQL with the psql-shell when writing SQL and managing the database, we also used pgAdmin as GUI for simplicity and to better understand what and how things worked. We started by as mentioned by the course material and the project page to create our OLAP queries. After creating the queries and running them on our database its important to be able to verify our output. To verify this we will do the calculations by hand to make sure we get the same result

Result

We start to create our queries we had to do some changes in the database and insert code to make all the queries work. All the changes can be found on our github at Github

The first query we constructed was to show the number of lessons given per month during a specified year. We constructed the query using postgreSQL what the code does it that we first select the month from the 'scheduled time slots' so that we can sum up all lessons and categorize them in months. We than count the the total number of lessons using an alias 'total_lessons'. We than use what is called a 'CASE' statement to add up all different type of lessons. After that we use 'UNION ALL' to combine them all. We then choose the year and group by month. the output we get when we use the current year 2022 can be seen in figure 1, you can also change to year to next year to see more lessons as the year is about to end.

	month numeric	total_lessons bigint	individual_lessons bigint	group_lessons bigint	ensemble_lessons bigint
1	12	13	5	5	3

Figure 1: number of lessons during a month

The next query we will show how many students there are with no sibling, with one sibling, with two siblings, etc. The query works as follows, we first select the students ids and how many siblings a student should have. We than start the sub query where we count number of siblings, we then group them up by number of siblings to show how many siblings each student has. In figure 2 we can see how the output looks when we execute the script.

	student_id integer	no_of_siblings bigint
1	4	1
2	5	1
3	1	2
4	2	2
5	3	2

Figure 2: number of siblings per student

The last manually executed query will be to list all instructors who has given more than a specific number of lessons during the current month. We start by selecting the instructors id and then sum up all different lesson types with a 'SUM' in a tables called lesson count. From there we then count all and after that extract the month from available time and also the current month. We do this for the three different types of lessons. Lastly we show the instructors that have a higher lesson count than our own value that we pick, which in this case was one just to check that everything worked we also tested with bigger values. Our output can be seen at figure 3.

The last query is a query that should be performed programmatically and there we query should list all ensembles held during the next week, sorted by music genre and weekday. For each ensemble tell whether it's full booked, has 1-2 seats left or has more seats left. What we do is we select all the information we need from the database and adds a new column for the weekdays. We than use cases as recommended in the assignment to make sure that we get the output correct depending on how many spots are filled. We then set an interval of seven days from the current day to make sure to only show the ensemble's taking place this coming week. Figure 4 is the output table for this query.

	instructor_id integer	total_lessons numeric
1	1	3
2	2	3
3	3	3
4	4	2
5	5	2

Figure 3: Instructors lessons

	id [PK] integer	scheduled_time_slot [PK] timestamp without time zone	target_genre character varying (500)	instructor_id [PK] integer	school_id [PK] integer	weekday numeric	seat_status text
1	1	2022-12-16 10:00:00	Classical	1	1	5	has more seats left
2	2	2022-12-18 11:00:00	Jazz	2	1	0	has more seats left
3	3	2022-12-20 12:00:00	Rock	3	2	2	has more seats left

Figure 4: List ensembles for a week

Discussion

Did you change the database design to simplify these queries? If so, was the database design worsened in any way just to make it easier to write these particular queries?

We had to do some changes to our database to make all the queries work. the one query that we had to make the most changes in our database was the one where we would list all siblings. To start with we had to add so that there was at least one sibling with one sibling one with 2 siblings and one without. We later found out that we also wasnt able to use the sibling relation that we hade used. This was because there was no way to find out who was sibling to who. To make this work we had to make changes on how our sibling table worked so that we had a relation between 2 students that showed that the where siblings.

Is there any correlated subquery, that is a subquery using values from the outer query? Remember that correlated subqueries are slow since they are evaluated once for each row processed in the outer query

?

Are there unnecessarily long and complicated queries? Are you for example using a UNION clause where it's not required?

This is most likely possible, do to the fact that this is the first time working in SQL i think its very possible that we have done written queries in a more complicated way than necessary. I do not think they are way to long but could mot likey be shorten down with a better understandning of SQL.

Analyze the query plan for at least one of your queries using the command EX- PLAIN (or EXPLAIN ANALYZE), which is available in both Postgres and MySQL. Where in the query does the DBMS spend most time? Is that

reasonable?

When we run an 'EXPLAIN ANALYZE' query like this we get a sort of rapport where we can find out where in the query we spend the most time and how costly it is. We can see that "cost" is a variable for how long the query takes to execute. But its important to note that its not the exact execution time rather an indication on how long it should take based on the time. Its always good to compare different costs to make sure you find issues with the time. We did this 'EXPLAIN ANALYZE' on the forth query and got the following output as seen in figure 5. We can se that row 1 is is the row with the biggest "cost".

	QUERY PLAN text
1	Sort (cost=13.17..13.18 rows=1 width=600)
2	Sort Key: target_genre, (EXTRACT(dow FROM scheduled_time_slot))
3	-> Seq Scan on ensemble (cost=0.00..13.16 rows=1 width=600)
4	Filter: ((scheduled_time_slot >= CURRENT_TIMESTAMP) AND (scheduled_time_slot <= (CURRENT_TIMESTAMP + '7 days'::interval)))

Figure 5: EXPLAIN ANALYZE query