Task 3 SQL

IV1351 Data Storage Paradigms

Klara Lindemalm, klindema@kth.se

Fall 2022

Introduction

For this task we intended to start using the database to see that all information is available and all data could be derived to fulfill the project description of the Sound Good Music School. Collaborator for this project was Rasmuss Per Brodin.

Literature Study

In order to prepare for this task we attended the live lecture on SQL held by Paris Carbone in KTH Kista. A glance through the pages in the book including chapter 6 and 7 from the main course book along with chapter 3 and 4 from the alternative book. Furthermore a brief reading though pages regarding relational calculus chapter 6.6 and 8.7 in the main text book and chapter 27 from additional course book. The document tips-and-tricks-task3.pdf was looked at along with the task 3 description provided on the canvas page for the project. The tutorial session provided for this task came to be very helpful when we got stuck with the SQL queries. Furthermore a good number of google searches to understand the SQL language fully and understand the errors that we stumbled upon when setting up the queries. A lot of this information was found on the postgres website along with an understanding of the explain analyze tool used in for the evaluation part of the queries.

Method

We where requested to set up four queries, description of each is found on the canvas page for the project task description:

1. lessons_per_month
2. student_sibling_count
3. instructor_lesson_count

4. ensembles_held_next_week

The database management system used was Postgres SQL run via Docker Management window. The code for the database, insert data and queries where written in the code editor Visual Studio Code and tested via the terminal window. To create more insert data we used the online generating tool. For the set up of the queries we used the following steps:

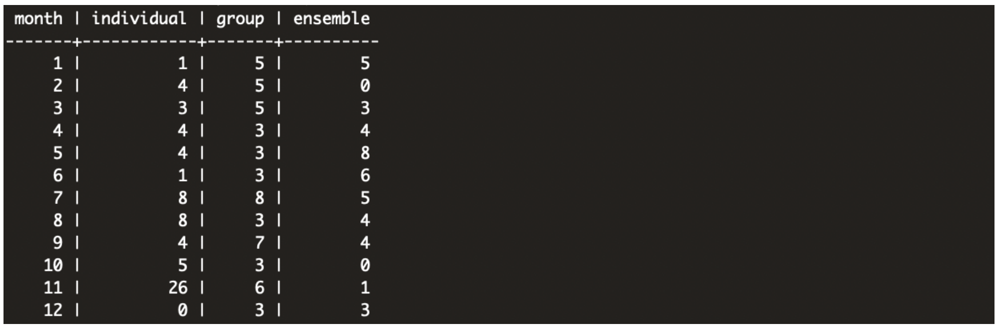
1. Decide what data needs to be selected.
2. Determine FROM where the selected data is found, in what table.
3. What requirements and conditions are requested from the query description.
4. Review the query.
5. Decide what type of view used for the queries, materialized or not.

The first step was to decide what data needs to be selected to preform the query, requested by the instructions given in the task description. In some cases the selected data has to be derived from the current data to be presented in the right format. The second step is to determine from where this data is to be selected. If only present in one table this step is very straight forward but in some cases one has to considers selecting data from several tables and how to join the data in the right full manner for example with an inner join. For the third step we decided what conditions and requirements are requested. In what order should the data be presented, are there only some of the data given in the table that should be presented in the query such as time limitations and other conditions. All mentioned above has to be reviewed in the forth step so that no information is missing and so that the presented data is altered in the rightful format requested bu the description of the query. Lastly we considered what view to use. Relevant questions for this step was to evaluate how often the query would be used? How often it would be updated? Lastly once all queries where written and the view was set we used the explain analyze tool to evaluate the efficiency of the query.

Result

First query

For the first query step two was easy as all data required could be found in table lesson. We selected month from the time_start column and selected a sum of all lesson_types with a case statement for each type individual, group and advanced. The sum function returns the total sum of a numeric column where the case statement is fulfilled. If fulfilled the case is set to 1 and if not set to 0. The conditional statement was that only the sum was selected for all lessons held in a given year (we choose 2022). Then all selected data was grouped and ordered by the selected months. This view was set to a materialized view.

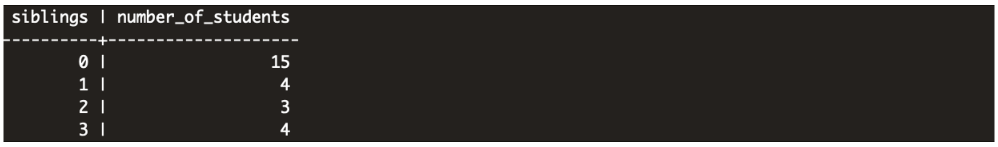


```
month | individual | group | ensemble
-------+------------+-------+----------
     1 |          1 |     5 |        5
     2 |          4 |     5 |        0
     3 |          3 |     5 |        3
     4 |          4 |     3 |        4
     5 |          4 |     3 |        8
     6 |          1 |     3 |        6
     7 |          8 |     8 |        5
     8 |          8 |     3 |        4
     9 |          4 |     7 |        4
    10 |          5 |     3 |        0
    11 |         26 |     6 |        1
    12 |          0 |     3 |        3
```

Figure 1: lessons_per_month

Second query

For the second query all data required could be selected from the student table. The data selected was siblings and number_of_students. This came to be a bit more complex as we needed to create a count for both the students with a family id and those that had a family id set to null, as these can't be altered from the same count function. In order to count all student who have no family id, we used the COUNT(*) function with a WHERE clause, where the family_id is set to null and set up a union with the result where students had a family_id. For the student who had a family id, we set up a inner select count statement as siblings, from student to select the number of students with the same family_id as siblings. This result was subtracted by one in or to not include the student in the siblings count result. This was then group by siblings to select the rows with the same value into one row. To select the number of student with the same number of siblings registered at the school we selected a count of the siblings table multiplied with siblings plus one as we subtracted one in the inner query. This column was named number_of_students. Lastly we set up the condition to order the result in ascending order to list the number of students with no siblings first. For the view we decided to set this one to a materialized view similar to the first view.

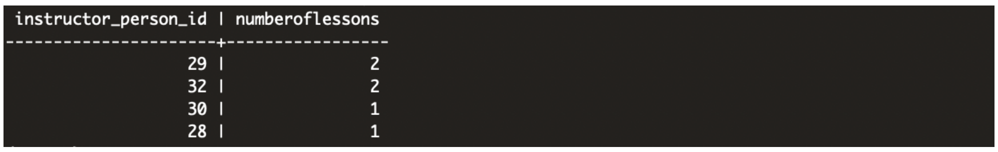


```
siblings | number_of_students
----------+--------------------
        0 |                 15
        1 |                  4
        2 |                  3
        3 |                  4
```

Figure 2: student_sibling_count

Third query

For the third query we selected the instructor_person_id and a count of all lesson_id as number_Of_Lessons. In the inner select statement where we decide from where the data can be fetched we selected distinct values of lesson_id from booking, instructor_person_id and the time_start variable from lesson. The data had to be fetched from two tables booking and lesson as the instructor can only be found in booking but a lesson can relate to several bookings if it is for example a group lesson. Therefore we used an inner join to select the lesson_id from booking where it was equal to the lesson_id in lesson as instructor_lesson. The conditions required that we extracted the current month from the current year, this was done with an AND. And lastly we ordered the query in descending order so that the instructors most likely to overwork themselves would be presented on the top. This query was to be executed daily therefore we choose to implement it as a materialized view.

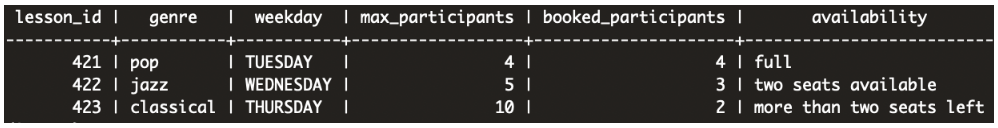


```
instructor_person_id | numberoflessons
---------------------+-----------------
                  29 |               2
                  32 |               2
                  30 |               1
                  28 |               1
```

Figure 3: instructor_lesson_count

Forth query

For the last query we selected lesson_id, genre, the day from time_start as weekday, max_participants and lastly booked_participants as a derived variable. Similar to the third query we collected the data from both booking and lesson with an inner join where the lesson type was ensemble and time_start stamp was between the current_week+ 7 days and plus 14 days altering the upcoming week. We crated a case with tree statements when the difference between max_participants and booked_participants was 0, one and 2 ending with and else statement when there are more seats than two available. We choose a regular view for this query.



```
 lesson_id |   genre   |  weekday  | max_participants | booked_participants |       availability
-----------+-----------+-----------+------------------+---------------------+--------------------------
       421 | pop       | TUESDAY   |                4 |                   4 | full
       422 | jazz      | WEDNESDAY |                5 |                   3 | two seats available
       423 | classical | THURSDAY  |               10 |                   2 | more than two seats left
```

Figure 4: ensembles_held_next_week

All queries can be found in the *link*.

If not possible to open the link type:
https://github.com/KlaraLindemalm/Soundgood-Music-School-Task-3

Discussion

When deciding the type of view to be used we evaluated mainly how often the query would be used. The tree first query's where to be used several times per week and more often the third query would be executed several times per day. To increase the speed to run the query we decided to set all queries in a materialized view. This is more costly but improves runtime significantly as the table is stored in the memory. This is of course something to be discussed with the customer as it alters a cost but in this case when we are the customer we decided that this was relevant for all three queries. The forth query isn't specified how often it will be used therefore we decided to not set it to a materialized view. The decision for this was evaluated and proven using the explain analyze tool of the third query instructor_lesson_count.

Type of query	**Planning Time**(ms)	**Excecution Time**(ms)
Creating the materialized view	3.002	18.817
Running the materialized view	0.156	0.136
Running without a view	3.171	3.011

Table 1: EXPLAIN ANALYZE of instructor_lesson_count, time in ms

Just creating the materialized view provides a greater cost than just running the query but when we have a materialized view and just run the query by the below statement runtime decreased significantly as displayed in Table 1. As we expect to have the table pre-built we won't have to create the table every-time the query is executed.

```
SELECT * FROM instructor_lesson_count;
```

One downside of a material view is that when used in some data-base systems they have to be updated manually when data is added to the data base relevant to the query. This is the case for PostgresSQL and the administrative staff of the Soundgood music school have to be aware of this to make sure that the queries are updated with relevant information. For a regular view there is no need to refresh the information as the query is set up every time we call the view.

```
REFERESH MATERIALIZED VIEW lessons_per_month;
```