The sound good music school - task 3

Rozhan Asadi

Fall term 2022

Introduction

In the first task a conceptual model was made for the sound good music school and that was the first step for making the database. In the last task, we continued the project and made a logical model for the data base while making sure the model is normalized. After the logical model was made, a SQL code was written to make the tables and the relations that we have in the database. Then some data was inserted into the database so as we get started with the database. In this task of the assignment we will be making online analytical processing queries and views. The method will be explained and the results will be shown at the end. This project is done only by the writer of this report, Rozhan Asadi.

Literature study

To do this task, I studied the lecture about SQL again and got help from the recommended book for this course. Additionally, when I faced some problems or questions in the middle of making the queries, I searched and got help from websites such as "w3schools.com".

Method

The SQL queries that were made in this task, are meant to manage and help the tasks that should be performed on the database. They help us analyze the business and create reports. Four different main tasks had to be done in this assignment.

Task 1: a query needs to be made to show the number of lessons given per month during a specified year

Task 2: a query needs to be made to show the number of siblings of the students

Task 3: a query needs to be made to list which instructors have had lessons more than a specific number

Task 4: a query needs to be made to list all ensembles that will be happening in the following

For designing the queries, postgreSQL to manage the system. PgAdmin is used to help postgreSQL to run the queries. Each query was checked manually to make sure that it works correctly.

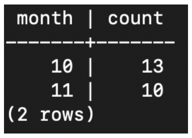
Results

In this section the results will be shown and discussed. To see the code clearly you can check out my Git repository: Click Link.

For the task one the following query was written:

```
create view counting_lessons_per_month as
select extract(month from date)as month,
count(*)
from Lesson
where extract(year from date)='2022'
group by extract(month from date)
order by extract(month from date) ASC;
```

In this view we can see the number of lessons in each month in the specific year '2022'.



```
 month | count
-------+-------
    10 |    13
    11 |    10
(2 rows)
```

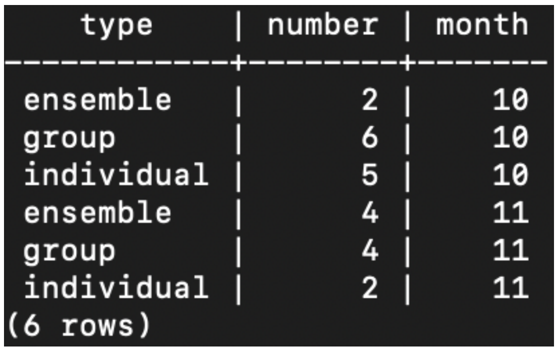
Figure 1: task one view

Part two:

```
create view counting_lessons_per_month_type as
select type,count(*) as number,
extract(month from date) as month
from lesson
where extract(year from date)='2022'
group by month,type;
```

Moving one to the next task, making a query that counts the siblings for students :

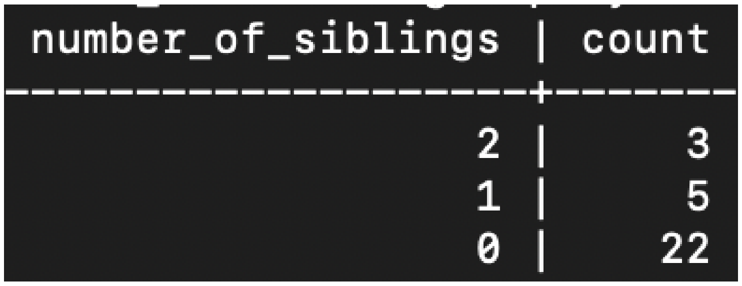
```
create view counting_siblings as
select sibling_count as number_of_siblings, count(sibling_count) as count
from (
select id, case
```



```
   type     | number | month
------------+--------+-------
 ensemble   |      2 |    10
 group      |      6 |    10
 individual |      5 |    10
 ensemble   |      4 |    11
 group      |      4 |    11
 individual |      2 |    11
(6 rows)
```

Figure 2: task one view - part 2

```
when hassibling=FALSE then 0
when (count(id)=1 and hassibling=TRUE) then 1
when (count(id)=2 and hassibling=TRUE) then 2 end
as sibling_count from student_sibling_info group by id,hassibling
) as subquery
group by sibling_count order by sibling_count desc;
```



```
 number_of_siblings | count
--------------------+-------
                  2 |     3
                  1 |     5
                  0 |    22
```

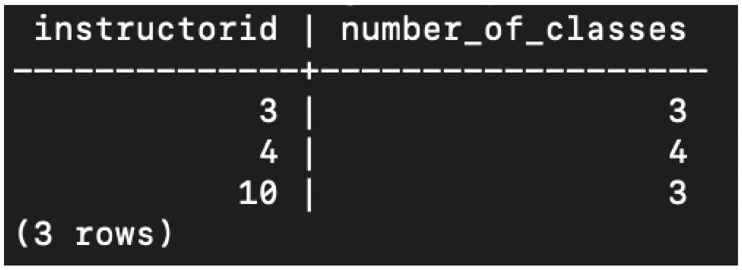
Figure 3: task two view

The third task was to make a query that helps the managers to watch out for risk of instructors over working and making themselves exhausted.

```
create view instructors_at_risk as
select instructorid,
count(instructorid) as number_of_classes
from lesson
group by instructorid
having count(instructorid)>2;
```

In the following figure we can see the id for the instructors that worked more than 2 times :

For the last task a query had to be made to see the ensembles of the schedule for the following week.



```
 instructorid | number_of_classes
--------------+-------------------
            3 |                 3
            4 |                 4
           10 |                 3
(3 rows)
```

Figure 4: task three view

```
create materialized view ensemble_weekly_schedule as
select lessonid,
genre,
to_char(date,’Day’) as weekday,
date,
case
when current_students=maxseats then ’the ensemble is full’
when current_students=maxseats-1 then ’1 seat left’
when current_students=maxseats-2 then ’2 seats left’
else ’more than 2 seats left’
end as empty_seats
from ensemble where
date_trunc(’week’,date)=date_trunc(’week’,now()) + interval ’1 week’
order by genre,weekday;
```

Discussion

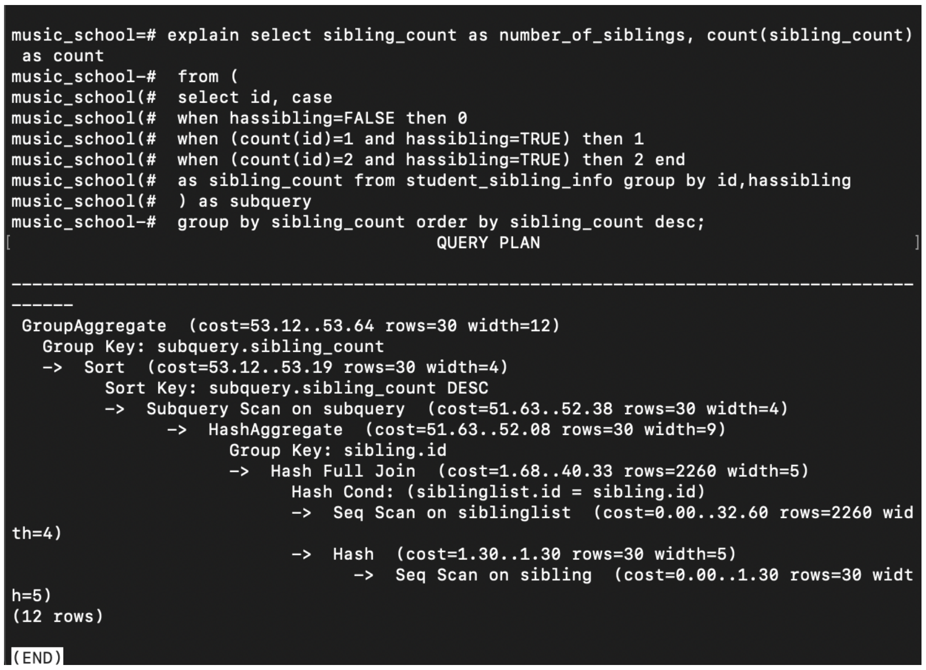
In some parts of the query making in the task, it was needed to make some changes to the tables or make new tables, For example for the task two, finding the siblings, it was needed to make a new table while joining two tables.

The queries were tried to be as short as possible so that it would be easy to compile and easy to understand. It was tried to keep everything without complication. The analyze the task two of this seminar can be seen below:

As it can be seen that first 3 tasks were done by creating a view but the last task was done by creating a materialized view. The reason is that materialized view gets stored at the hard disk and is a physical copy of the table that we are creating. Since the last task is executed manually daily, it is better to use a materialized view which is fast processing too in contrast to a view which is slow processing.

Historical data database

A historical data database is data collected about past events in the main database. Historical databases are typically used to store large amounts of



```
music_school=# explain select sibling_count as number_of_siblings, count(sibling_count)
 as count
music_school-#  from (
music_school(#  select id, case
music_school(#  when hassibling=FALSE then 0
music_school(#  when (count(id)=1 and hassibling=TRUE) then 1
music_school(#  when (count(id)=2 and hassibling=TRUE) then 2 end
music_school(#  as sibling_count from student_sibling_info group by id,hassibling
music_school(#  ) as subquery
music_school-#  group by sibling_count order by sibling_count desc;
[                                       QUERY PLAN                                      ]

-------------------------------------------------------------------------------------
------
 GroupAggregate  (cost=53.12..53.64 rows=30 width=12)
   Group Key: subquery.sibling_count
   ->  Sort  (cost=53.12..53.19 rows=30 width=4)
         Sort Key: subquery.sibling_count DESC
         ->  Subquery Scan on subquery  (cost=51.63..52.38 rows=30 width=4)
               ->  HashAggregate  (cost=51.63..52.08 rows=30 width=9)
                     Group Key: sibling.id
                     ->  Hash Full Join  (cost=1.68..40.33 rows=2260 width=5)
                           Hash Cond: (siblinglist.id = sibling.id)
                           ->  Seq Scan on siblinglist  (cost=0.00..32.60 rows=2260 wid
th=4)
                           ->  Hash  (cost=1.30..1.30 rows=30 width=5)
                                 ->  Seq Scan on sibling  (cost=0.00..1.30 rows=30 widt
h=5)
(12 rows)

(END)
```

Figure 5: analysis of task 2

data that can be easily queried and analyzed. In our case, what we want to back up and store is the lessons and the students that take each lesson while also showing what is the type of each lesson. The historical data database that we need was made in the following way:

```
 select * into lesson_historical_db from
(select * from lessons full join lesson_person
on lesson.lessonid = lesson_person.lessonid) as subquery;
```

The lesson entity was created in a way that contained attributes price and lesson type. Having that, we can join lessons with lesson-person (which contains the students and the lessons that each are taking) and get all the info that we need.

Now that we have talked about historical databases, let's discuss the advantages and disadvantages of denormalization. As you know, the process of adding redundant data to a database in order to improve query performance is called denormalization. Denormalization improves the query performance and simplifies the schema so the model will be easier to understand. However, denormalizing a database increases the disk usage which is not really optimal. It also reduces the data scalability and consistancy. By denormalizing a database, there is a possibility to lose the power of database constraints and increase the risk of data inconsistencies if not done properly.