Logical Model

Data Storage Paradigms, IV1351

Robert Furuvald, rfu@kth.se

2023-01-10

1 Introduction

This task is to create different queries to test the database designed in the previous assignment.

For this task the author watched the lectures given about SQL and read chapter 6 and 7 I have worked on my own.

2 Method

The different queries to create and test on the database was

1. Show number of lessons given per month during a specific year
2. Show how many students with no sibling, one sibling, two siblings
3. List all instructors who have given more than a specific number of lessons during the current month
4. List all ensambles held during the next week.

For this assignment the author has used PostgreSQL and pgAdmin 4 as Database Management. The data is generated into a script that is runned from pgAdmin. As for verifying that the SQL queries work as intended, one common method is to use the SELECT statement to retrieve data from the tables and check that the results match the expected output. Another method would be to use the INSERT, UPDATE and DELETE statements to add, modify, and remove data from the tables, and then use the SELECT statement to check that the changes were made correctly. The author manually verified the queries from what data was inserted.

3 Result

Git where the scripts can be found for creating and inserting fake data and the different queries https://github.com/RobertFuruvald/IV1351.

The first query was to show number of lessons given per month during a specific year. Query is shown by figure 3.1.

```
--number of lessons a certain month
CREATE VIEW lesson_count_type AS
SELECT
    COUNT(*) as total,
    COUNT(*) FILTER(WHERE price_id=1 or price_id=2 or price_id=3) as individual,
    COUNT(*) FILTER(WHERE price_id=4 or price_id=5 or price_id=6) as group,
    COUNT(*) FILTER(WHERE price_id=7) as ensamble
    FROM public.lesson
    WHERE EXTRACT(YEAR FROM lesson_date)=EXTRACT(YEAR FROM CURRENT_DATE)
    GROUP BY EXTRACT(MONTH FROM lesson_date);
```

Figure 3. 1: Displays the first query

The result was verified manually by checking number of lessons for that month using pgAdmin 4. Result is shown by figure 3.2.

	total bigint	individual bigint	group bigint	ensamble bigint
1	196	64	112	20
2	164	64	80	20

Figure 3. 2: Displays the result for the first query

Second query shows how many students that have zero, one or two siblings, which is shown by figure 3.3.

```
--number of siblings
CREATE VIEW number_of_siblings AS
SELECT COUNT(student_id) as student_count,
      CASE
        WHEN siblings = 0 THEN 'No siblings'
        WHEN siblings = 1 THEN 'One sibling'
        WHEN siblings = 2 THEN 'Two siblings'
        ELSE 'More than two siblings'
      END as sibling_group
FROM (SELECT student_id,
            (SELECT COUNT(*) FROM student_siblings WHERE student_id = s.student_id) as siblings
      FROM student s) as students_with_siblings
GROUP BY siblings
ORDER BY siblings;
```

Figure 3. 3: Displays the second query

The result was verified manually by checking number of lessons for that month using pgAdmin 4. Result is shown by figure 3.4.

	student_count bigint	sibling_group text
1	29	No siblings
2	16	One sibling
3	15	Two siblings

Figure 3. 4: Displays the result of the second query

For this query it list all instructors who has more than one lesson this month and views the total number of lessons per instructor.

```
-- number of lessons per instructor
CREATE VIEW lesson_count_instructor as
SELECT
    public.person.first_name, public.person.last_name, public.instructor.instructor_id,
    COUNT(*) FILTER (WHERE EXTRACT(MONTH FROM lesson_date) = EXTRACT(MONTH FROM CURRENT_DATE)) as numoflessons
    FROM lesson
    INNER JOIN public.instructor ON  public.lesson.instructor_id = public.instructor.instructor_id
    INNER JOIN public.person ON public.instructor.person_id = public.person.person_id
    GROUP BY person.first_name, person.last_name, instructor.instructor_id
    HAVING COUNT(*) FILTER (WHERE EXTRACT(MONTH FROM lesson_date) = EXTRACT(MONTH FROM CURRENT_DATE)) > 1
    ORDER BY numoflessons ASC;
```

Figure 3. 5: Displays the third query.

The result was verified manually by checking number of lessons per instructor using pgAdmin 4. Result is shown by figure 3.6.

	first_name character varying (50)	last_name character varying (50)	instructor_id integer	numoflessons bigint
1	Meghan	Wilkinson	6	5
2	Gil	Rodriquez	3	15
3	Warren	Aguilar	7	19
4	Mara	Beard	4	19
5	Pamela	Sears	5	24
6	Wendy	Russo	2	26
7	Jesse	Garner	8	37
8	Austin	Hood	1	51

Figure 3. 6: Displays the result of the third query.

For this query it list all lessons which is an ensamble for a period of a week and prints how many seats every lesson have left, it sorts it by genre and date. The query is shown in figure 3.7 and the result in figure 3.8.

```
--ensambles for next week with available room left
CREATE MATERIALIZED VIEW lessons_next_week AS
SELECT lesson.lesson_date,genre.genre,group_lesson.max_students, COUNT(lesson_attendees.lesson_id) as enrolled,
  CASE
    WHEN COUNT(student_id) = CAST(group_lesson.max_students as INT) THEN 'Full'
    WHEN COUNT(student_id) >= CAST(group_lesson.max_students AS INT) - 2 THEN '1-2 seats left'
    ELSE 'More seats left'
  END AS availability
    FROM public.lesson_attendees
    inner join lesson ON lesson_attendees.lesson_id = lesson.lesson_id
    inner JOIN group_lesson ON lesson_attendees.lesson_id = group_lesson.lesson_id
    inner join ensamble ON ensamble.lesson_id = lesson_attendees.lesson_id
    inner join genre ON genre.genre_id=ensamble.genre_id
    WHERE EXTRACT(WEEK FROM lesson.lesson_date)=EXTRACT(WEEK FROM CURRENT_DATE+7)
    GROUP BY genre.genre, group_lesson.max_students, lesson_attendees.lesson_id, lesson.lesson_date
    ORDER BY genre.genre ASC, lesson.lesson_date ASC;
```

Figure 3.7: Displays the fourth query

	lesson_date date	genre character varying (50)	max_students character varying (50)	enrolled bigint	availability text
1	2023-01-20	classic	7	5	1-2 seats left
2	2023-01-20	country	7	5	1-2 seats left
3	2023-01-20	jazz	7	5	1-2 seats left
4	2023-01-20	metal	7	4	More seats left
5	2023-01-20	rock	7	4	More seats left

Figure 3.8: Displays the result of the fourth query

4 Discussion

A view is a virtual table that is based on the result of a SELECT statement. A view can be thought of as a saved query that can be used like a table in other SQL statements. Views can be used to simplify complex queries by encapsulating the logic of the underlying query and exposing only the relevant columns to the application or user. They also can be used for security purposes, for example, by providing a limited view of the data to specific users or applications.

Materialized views, on the other hand, are a pre-calculated, stored version of a view, that can be used to speed up read-only queries, instead of executing the query every time it is requested. Because the data is stored in the view, it can be queried directly, rather than recalculating the query each time.

It depends on the usage and query patterns, if views and materialized views would benefit a certain query or not. It is important to consider the trade-offs between the performance benefits of using a materialized view versus the cost of maintaining it, as the underlying data can change and the view needs to be updated.

For these queries I made all as normal views except the last query which where made as a materialized view. This was chosen because lessons will many people want to access like students and teachers and also other staff members.

I did not change the database for these queries.