Seminar 5
Data Storage Paradigms, IV1351

Dipsikha (Diddi) Dutta,
dipsikha@kth.se

6/1 2024

Contents

1 Introduction

The task of this seminar is to formulate queries such that it fulfills to provide the required data for different analysis questions. The aim is to create OLAP queries and views to serve to analyze the business and to create reports, specifically for the requests *The Soundgood Music School* have to make. I had little time left to do this assignment because of the other course, so I worked by myself, otherwise I would collaborate with some others. The database system I used for this assignment is PostgreSQL. The data that is needed to execute and get reports by the queries were made in seminar 2.

2 Literature Study

Before beginning the assignment I went through the whole video material for the SQL language and realized it would have been good if I went through the video before creating the logical model in seminar 2. This is because it would be a bit easier to understand how queries would be formulated but I personally still find it hard to wrap my head around making attributes with cardinality more than one as a separate entity, which was a guideline for making the logical model. Moreover the SQL tutorial video from the course I also searched up external video sources on youtube on how to work with scripts specifically for postgre and looking up usable clauses and functions like DOW, INTERVAL etc... I also practiced queries in with this pedagogical and helpful tutorial website: https://sqlbolt.com/

3 Method

I used pg-admin tool for developing the SQL queries in postgre. I verified the scripts by getting relevant tables as return of those queries.

4 Result for seminar 5

https://github.com/Diddi25/Datalagring

The sql scripts can be downloaded from the directory **sem5** in the github link above. The scripts contain both the result table and the script files. The result tables were copypasted from the SQL shell terminal. All the text in this section is updated for seminar 5. Since the remake of the logical model in seminar 2, I also had to modify all the queries for this task.

The feedback from seminar 3 was the following:

1. The github link does not contain query for task3.
Now it does.

2. Query 1 is incorrect, It does not show the specific number of individual lessons, group lessons and ensembles
Now it does show for the specific lesson types.

Query 1: Total number of lessons per month in a year

```
SELECT
    TO_CHAR(start_time, 'Mon') AS Month,
    COUNT(*) AS Total,
    COUNT(*) FILTER (WHERE lesson_type_id = 3) AS Individual,
    COUNT(*) FILTER (WHERE lesson_type_id = 1) AS Group,
    COUNT(*) FILTER (WHERE lesson_type_id = 2) AS Ensemble
FROM
    frukt.lesson
WHERE
    EXTRACT(YEAR FROM start_time) = 2023
GROUP BY
    Month
ORDER BY
    month;
```

Output:

```
month|total|individual|group|ensemble|
-----+-----+----------+-----+--------+
Dec  |   11|         0|    7|       7|
Nov  |    3|         0|    2|       1|
```

With the EXPLAIN ANALYZE query:

```
QUERY PLAN                                                                                    |
----------------------------------------------------------------------------------------------+
GroupAggregate  (cost=2.27..2.30 rows=1 width=64) (actual time=0.485..0.489
rows=2 loops=1)             |
  Group Key: (to_char(start_time, 'Mon'::text))                                               |
  ->  Sort  (cost=2.27..2.27 rows=1 width=36) (actual time=0.467..0.468 rows=17
loops=1)                |
        Sort Key: (to_char(start_time, 'Mon'::text))                                          |
        Sort Method: quicksort  Memory: 25kB                                                  |
        ->  Seq Scan on lesson  (cost=0.00..2.26 rows=1 width=36) (actual
time=0.389..0.422 rows=17 loops=1)|
              Filter: (EXTRACT(year FROM start_time) = '2023'::numeric)
|
Planning Time: 0.744 ms                                                                       |
Execution Time: 0.541 ms                                                                      |
```

The relevant tables needed for this query was the “lesson” and “lesson_type” relations. Since the start_time and end_time attributes in “lesson” was specified to by of attribute type TIMESTAMP I extracted the “Year” and “Mon” parts as Year and month making it easy to present the result.

By analyzing the EXPLAIN ANALYZE output from this query it seems that the conclusion is that the query is pretty efficient by the overall execution time. It can be read that the machine used the “quick-sort” method and was memory efficient. Usually the sequential scan is more costly compared to index scanning, but in this case is not needed since the data amount is small.

Query 2: Total number of siblings for a number of students

```
CREATE OR REPLACE VIEW SiblingCount AS
    SELECT student_id, COUNT(sibling_id) AS count_siblings
    FROM frukt.sibling
    GROUP BY student_id;

SELECT
    n AS "No of Siblings",
    COUNT(student_id) AS "No of Students"
FROM
    generate_series(0, 2) n
LEFT JOIN
    SiblingCount sc ON n = sc.count_siblings
GROUP BY
    n
ORDER BY
    n;
```

```
No of Siblings|No of Students|
--------------+--------------+
             0|             0|
             1|             1|
             2|             1|
```

The relevant tables needed for this query was the “student” and “sibling” entity. To begin with only extracting the student id:s and count the number of sibling tuples that were made in a separate view. Using that to further create a series from 0-2 by the number of siblings, and do a left join with that view made it possible to retrieve the desired output result tables.

Query 3: Total number of lessons given per teacher

```
SELECT
   i.instructor_id AS "Instructor Id",
   p.name AS "Name",
   COUNT(l.lesson_id) AS "No of Lessons"
FROM
   frukt.instructor i
JOIN
   frukt.lesson l ON i.instructor_id = l.instructor_id
JOIN
   frukt.person p ON i.person_id = p.person_id
WHERE
   DATE_TRUNC('MONTH', l.start_time) = '2023-12-01'::date
GROUP BY
   i.instructor_id, p.name
HAVING
   COUNT(l.lesson_id) > 3
ORDER BY
   COUNT(l.lesson_id) DESC;
```

```
Instructor Id|Name         |No of Lessons|
-------------+-------------+-------------+
            1|John Doe     |            5|
            2|Jane Smith   |            5|
            3|Alice Johnson|            4|
```

The relevant tables for this query was the “instructor”, “lesson” and “person” entities. The person relation were needed to extract the names for each instructor. By joining the relevant tables, counting the number of lessons some instructors has given (by searching with their id) and having more than 3 lessons given from the start date, extracted with DATE_TRUNC, the result was the values in the columns above. Since the start_time attribute is of type TIMESTAMP I had to extract only the date part with the double colon syntax ‘::’.

Query 4: List all ensembles held during the next week

```
CREATE OR REPLACE VIEW EnsembleAvailability AS
  SELECT
    l.lesson_id,
    l.start_time,
    l.genre,
    l.max_students,
    COUNT(*) AS enrolled_students,
    CASE
      WHEN COUNT(*) = l.max_students THEN 'Full Booked'
      WHEN COUNT(*) >= l.max_students - 2 THEN '1-2 Seats Left'
      ELSE 'Many seats'
    END AS availability
  FROM
    public.lesson l
  WHERE
    l.lesson_type_id = 2
    AND l.start_time >= NOW()::date
    AND l.start_time < NOW()::date + interval '1 week'
  GROUP BY
    l.lesson_id, l.start_time, l.genre, l.max_students;

SELECT
  e.lesson_id,
  e.start_time,
  e.genre,
  e.availability,
  TO_CHAR(e.start_time, 'Day') AS weekday
FROM
  EnsembleAvailability e
ORDER BY
  e.genre, e.start_time, e.availability;
```

```
lesson_id|start_time             |genre  |availability  |weekday  |
---------+-----------------------+-------+--------------+---------+
       16|2024-01-09 14:00:00.000|Indie  |Many seats|Tuesday  |
       17|2024-01-11 18:00:00.000|Disco  |Many seats|Thursday |
```

The relevant tables for this query was mainly the "lesson" and "lesson_type" relations. The time components were a part of the lesson entity compared to my earlier attempt which made the query somewhat less complex. The first view is to check how many students there were left from a booking, with the help of the attribute “max_students” in lesson, providing all the cases for the CASE statement. By using the “INTERVAL” and now() function I could retrieve the bookings for next week, which is done in the WHERE clause. The WHERE clause also contains the lesson_type_id which specifies that it is lesson type “ensemble” it is desired to retrieve.

5 Discussion

- **Are views and materialized views used in all queries that benefit from using them? Can any query be made easier to understand by storing part of it in a view? Can performance be improved by using a materialized view?**

I learnt about the difference between serial and materialized views after making this assignment which made me to only create views, making the query in different parts. However, by learning about the materialized view later I realized that this type of view has better performance, since retrieving and updating relevant data in the query, in comparison to the serial view that only invokes the parent query. The first query may have been a bit too partialized and could be in some of the parts together, however, since it was the first query to work with, I retrieved relevant tables step by step, making the subqueries as part of the view, invoking each other in the hierarchy, making it more controllable and understandable.

- **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?**

I needed to change some relationships and add some attributes as foreign keys in other entities to make the queries a bit simplified. For example, to count all the lessons given during a specific year, the time-specific data was needed which were only available in the “bookings” entity. To solve this issue, a one-to-many relationship was created between “lesson” and “bookings” with the foreign key “booking_id” in lesson as an attribute. The database design did not get necessarily worse by doing this, however, when making the queries the logical model could have been more developed since getting more familiar with queries.

- **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.**

I did not use any query inside of another query, however making use of creating views. This might have caused the subqueries to be slow because of using values from the parent queries in the views. However, while making use of materialized views might have enhanced the performance of the queries, the queries are a bit complicated to formulate for retrieving the desired report tables.

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

The UNION clause is not used in the queries, however, the queries might have gotten a bit too complicated with the many entities made in seminar 2. This could have been done better if the “instructor” entity could have a foreign key attribute “lesson_id” to only make a count of the total number of lessons for each instructor. The same with “nr_of_students_in_lesson” entity where this could have been an attribute in “lesson” entity.

- **Analyze the query plan for at least one of your queries using the command EXPLAIN (or EXPLAIN ANALYZE), which is available in both Postgres and MySQL. Where in the query does the DBMS spend most time? Is that reasonable? If you have time, also consider if the query can be rewritten to execute faster, but you're not required to do that. The postgres documentation is found at https://www.postgresql.org/docs/current/using-explain.html and https://www.postgresql.org/docs/current/sql-explain.html There's also some explanation of EXPLAIN in the document Tips and Tricks for Project Task 3.**

The EXPLAIN ANALYZE function is used in the query for finding out how many siblings that groups of students have. In the query plan it is stating using a sequential scan on the "nr_of_siblings" entity. The query seems to take most time by doing that sequential scan since the estimated cost in between the parenthesis (cost=[...]) is relatively higher than the actual time. This execution difference is higher than the sort cost, which is specified in the query plan to have been used the QuickSort method to sort, which is a pretty fast algorithm. The memory cost is also relatively low in this case for sorting, 25kB. Overall it might seem reasonable, but I feel like I will get better at analyzing these queries over time since this is the only query I have ever analyzed for now.