SQL
Data Storage Paradigms, IV1351

Liam Battini
liamb@kth.se

1. Introduction
This report covers the sql queries used for gathering specific information from a database belonging to a music school company. The database is based of a logical-physical model created in the previous seminar where the database should be able to do all queries stated in the instructions.

2. Literature Study
Before working on the queries, I used my existing knowledge from the first lab in the course and a course I read in high school that consisted of some MySQL. But during the seminar when I got stuck trying to figure out how to do the queries, I gathered the needed knowledge through PostgreSQL's official documentation (PostgreSQL 15.1 Documentation, 2022).

3. Method
I used PostgreSQl as the DBMS which is the one used by the teacher in the course. Firstly, whenever I created a query, I made sure to join every table needed so I could be sure I had all the data I needed. Secondly, I filtered out the data with the where statement to make sure I only include the intended columns. Thirdly, I created the select columns and then my query was most likely done. However, I did not necessarily follow the three steps in order, I usually went forth and back between the steps.

When it came to verifying the data was accurate, I firstly made sure I had data so the query result would not be empty. Secondly, if I had any columns that counted something I made sure to calculate the data manually, so it adds up.

4. Result
Link to the sql files:
https://gits-15.sys.kth.se/liamb/IV1351/tree/main/sem2

```
music_school=# select * from lesson_per_month_amount;
 month | number_of_lessons
-------+-------------------
     1 |                 5
     2 |                 3
     3 |                 2
     4 |                 1
     5 |                 5
     6 |                 4
     7 |                 4
     8 |                 4
     9 |                 4
    10 |                 1
    11 |                 1
    12 |                 2
(12 rows)
```

Figure 4:1 Query 1.1 Number of lessons per month in specific year

In this query I selected year 2020 since that year consisted with the most amount of lessons in my database. I firstly extracted month from time in my lesson table in the month column. Secondly I extracted all the data from only year 2020 and lasted I grouped everything by month so it would show one month per row.

```
music_school=# select * from lesson_type_per_month_amount;
 month | number_of_lessons | ensemble | group_lesson | individual_lesson
-------+-------------------+----------+--------------+-------------------
     1 |                 5 |        0 |            0 |                 5
     2 |                 3 |        0 |            1 |                 2
     3 |                 2 |        0 |            1 |                 1
     4 |                 1 |        0 |            0 |                 1
     5 |                 5 |        1 |            1 |                 3
     6 |                 4 |        0 |            1 |                 3
     7 |                 4 |        0 |            0 |                 4
     8 |                 4 |        2 |            0 |                 2
     9 |                 4 |        2 |            0 |                 1
    10 |                 1 |        0 |            0 |                 1
    11 |                 1 |        0 |            0 |                 1
    12 |                 2 |        0 |            0 |                 2
(12 rows)
```

Figure 4:2 Query 1.2 Number of lessons per month by lesson type in specific year

This query is quite similar to the one used in Figure 4:1 besides I left joined every lesson type table and in select added a count with a case that sets it to 1 if the lesson_id from lesson is the same as in the lesson type lesson_id, otherwise it got NULL. And clearly the math is correct since the amount of lessons in the three different lesson type columns adds up to the number_of_lessons column.

```
music_school=# select * from student_sibling_amount;
 two_siblings | one_siblings | no_siblings
--------------+--------------+-------------
            3 |            4 |          18
(1 row)
```

Figure 4:3 Query 2 mount of siblings per student

This query was complicated where I had to create an alias column not shown in results that calculates the number of siblings each student_id had. Then once that was done it would go through the three different columns and increment the value depending on the number of siblings the student_id had inside the alias. The result is accurate since I know there are 25 students in total and I can easily see in the student_siblings table the data from the view is accurate.

```
music_school=# select * from instructor_lesson_per_month_amount;
 first_name | last_name | person_number | number_of_lessons
------------+-----------+---------------+-------------------
 Lillith    | Parrish   | 19535094      |                 2
 Marsden    | Fuentes   | 20004776      |                 2
 Aidan      | Kerr      | 19785823      |                 1
 Iliana     | Mcneil    | 19909757      |                 1
 Connor     | Hewitt    | 19922392      |                 1
(5 rows)
```

Figure 4:4 Query 3 Amount of lessons per instructor in specific month with a minimum amount of lessons

In this query I selected year 2020 month 10 (October) since from what I could notice that was the month in my database that consisted with the most amount of lessons meaning a more interesting result. This view was easy to create where I just had to gather some of the instructor's personal information and count its total amount of lessons which was easily gathered since we only gathered the rows where the lesson's instructor_id matched the instructor_id in the in the instructor table. Lastly the minimum amount of number_of_lessons had to be 0 in order to appear in the results and the data is shown depending on the amount of lessons from highest to lowest.

```
music_school=# select * from next_week_ensembles;
 lesson_id | genre | time | min_students | max_students | seats_taken | lesson_register_status
-----------+-------+------+--------------+--------------+-------------+-----------------------
```

Figure 4:5 Query 4.1 Ensembles next week

Since every lesson independent of lesson type is stored in the lesson table and there are no ensembles booked next week the results is empty. Therefore, In Figure 4:6 I demonstrated the query by selecting the ensembles between previous 500 days and next week.

```
music_school=# select * from next_week_ensembles_example;
 lesson_id |  genre  |        time         | min_students | max_students | seats_taken | lesson_register_status
-----------+---------+---------------------+--------------+--------------+-------------+-----------------------
        99 | Classic | 2022-10-30 15:23:00 |            4 |           11 |           9 | 1 or 2 seats left
        95 | Jazz    | 2022-08-18 14:30:00 |            5 |           13 |          10 | 3 or more seats left
        94 | Rock    | 2021-08-14 16:21:00 |            4 |           11 |          11 | Fully booked
(3 rows)
```

Figure 4:6 Query 4.2 Ensembles between previous 500 days and next week

I firstly calculated seats_taken from counting the the amount of rows in ensemble_students depending on the lesson_id. Afterwards to get the correct lesson_register_status I used case and calculated the difference between max_students and query used to get seats_taken. If the difference was 0, then it returned “Fully booked”, if it was between 1 or 2 then it returned “1 or 2 seats left” otherwise it would return “3 or more seats left”. When it came to selecting the time I used where and checked if the time is between current_date – 500 days and current_date + 7 days meaning it would check every ensemble in the previous 500 days and the following 7 days. Then I made sure to group by ensemble’s lesson_id, genre, time and lastly ordered by genre and time which is hard to tell since there are no common genres. But if you look at the genre table Classic has genre_id 5, Jazz id 3 and Rock 2 meaning they are ordered.

5. Discussion

View is used in this seminar where every query exists as views. Materialized view is also where it has been used for queries 4.1 and 4.2 shown in figures 4.5 and 4.6.

Before working on this seminar, I had to redo the previous seminar due to not meeting all the requirements. Whilst working on improving the logical-physical model of the database I thought of how I could do the queries for this seminar at the same time so I modelled it in a specific way that would not need to be remodeled for this seminar. I would say after updating the model of the database it made it easier to write the queries for this seminar since I already thought in my head which table joins, I would have to do to make sure I had all the necessary data.

Queries 4.1 and 4.2 could have been improved where I used COUNT(ensemble_students.lesson_id) 4 times and to simplify it I should have defined an alias for it and use it instead. I did try to improve it by using the column name seats_taken that was already defined but I was not able to use it. I still do not understand why I am not able to use sets_taken and I have not tried to figure it out due to laziness.

```
music_school=# explain analyze select EXTRACT(MONTH from time) as month, COUNT(*) as number_of_lessons, COUNT(CASE when(lesson.lesson_id = ensemble.lesson_id) then 1 else N
ULL END) as ensemble, COUNT(CASE when(lesson.lesson_id = group_lesson.lesson_id) then 1 else NULL END) as group_lesson, COUNT(CASE when(lesson.lesson_id = individual_lesson
.lesson_id) then 1 else NULL END) as individual_lesson from lesson left join ensemble on lesson.lesson_id = ensemble.lesson_id left join group_lesson on lesson.lesson_id =
group_lesson.lesson_id left join individual_lesson on lesson.lesson_id = individual_lesson.lesson_id where EXTRACT(year from time) = '2020' group by month order by month
music_school-# ;
                                                QUERY PLAN
--------------------------------------------------------------------------------------------------------------------------------------------
 Sort  (cost=36.51..36.51 rows=1 width=64) (actual time=0.293..0.295 rows=12 loops=1)
   Sort Key: (EXTRACT(month FROM lesson."time"))
   Sort Method: quicksort  Memory: 25kB
   ->  HashAggregate  (cost=36.48..36.50 rows=1 width=64) (actual time=0.274..0.279 rows=12 loops=1)
         Group Key: EXTRACT(month FROM lesson."time")
         Batches: 1  Memory Usage: 24kB
         ->  Nested Loop Left Join  (cost=2.84..29.88 rows=330 width=48) (actual time=0.108..0.249 rows=36 loops=1)
               ->  Nested Loop Left Join  (cost=2.68..12.73 rows=18 width=20) (actual time=0.098..0.173 rows=36 loops=1)                     ->  Hash Right Join  (cost=2.53
..4.55 rows=1 width=16) (actual time=0.086..0.115 rows=36 loops=1)
                           Hash Cond: (individual_lesson.lesson_id = lesson.lesson_id)
                           ->  Seq Scan on individual_lesson  (cost=0.00..1.80 rows=80 width=4) (actual time=0.008..0.014 rows=80 loops=1)
                           ->  Hash  (cost=2.51..2.51 rows=1 width=12) (actual time=0.068..0.068 rows=36 loops=1)
                                 Buckets: 1024  Batches: 1  Memory Usage: 10kB
                                 ->  Seq Scan on lesson  (cost=0.00..2.51 rows=1 width=12) (actual time=0.021..0.054 rows=36 loops=1)
                                       Filter: (EXTRACT(year FROM "time") = '2020'::numeric)
                                       Rows Removed by Filter: 65
                     ->  Index Only Scan using pk_ensemble on ensemble  (cost=0.15..8.17 rows=1 width=4) (actual time=0.001..0.001 rows=0 loops=36)
                           Index Cond: (lesson_id = lesson.lesson_id)
                           Heap Fetches: 5
               ->  Memoize  (cost=0.16..8.18 rows=1 width=4) (actual time=0.001..0.001 rows=0 loops=36)
                     Cache Key: lesson.lesson_id
                     Cache Mode: logical
                     Hits: 0  Misses: 36  Evictions: 0  Overflows: 0  Memory Usage: 3kB
                           Index Cond: (lesson_id = lesson.lesson_id)
                           Heap Fetches: 4
 Planning Time: 0.511 ms
 Execution Time: 0.374 ms
(28 rows)
```

Figure 5:1 Explain analyze of Query 1.2 shown in Figure 4:2.

This data is quite confusing for me, but I would assume the grouping the data by month takes most time since it goes from 0.105 – 0.249ms when it does the first nested left join to 0.274-0.279ms by grouping the data which is approximately a 0.1ms increase. Afterwards the two nested loop joins take the most time where the Seq scan and Hash takes together 0.076-0.082ms which is approximately a 0.05-0.1ms jump by just doing the two joins.

I think it makes sense grouping the data takes the most time since it must calculate so all data is shown based of the month. Then secondly it has to combine all the data with the joins which should also take some time since it goes through a loop where it checks a condition on every loop.

6. References

PostgreSQL 15.1 Documentation. (2022, 12 24). Retrieved from PostgresSQL: https://www.postgresql.org/docs/current/index.html