Project Report

Data Storage Paradigms, IV1351
Seminar 2 (Task 2 / 5.2)

Alexander Tashkinov

2026-01-06

Project members:
Alexander Tashkinov, atas@kth.se

GitHub repository link

Declaration

By submitting this assignment, it is hereby declared that all group members listed above have contributed to the solution. It is also declared that all project members fully understand all parts of the final solution and can explain it upon request. It is furthermore declared that the solution below is a contribution by the project members only, and specifically that no part of the solution has been copied from any other source (except for lecture slides at the course IV1351), no part of the solution has been provided by someone not listed as a project member above, and no part of the solution has been generated by a system.

1 Introduction

The goal of Seminar 2 (Task 2) was to create OLAP-style queries and views on top of the database designed in Seminar 1. The queries are intended for manual execution and should support reporting and analysis of planned workload, allocated workload, and teacher load distribution during study periods. In addition, at least one query must be analyzed using `EXPLAIN ANALYZE` to discuss performance and potential optimizations.

2 Literature Study

The work was prepared by reviewing course material on SQL, aggregation, grouping, joins, and views. Particular focus was placed on OLAP-style query patterns such as

pivoting activity rows into columns using `CASE` expressions, as well as PostgreSQL-specific performance analysis with `EXPLAIN (ANALYZE, BUFFERS)`.

3 Method

The database used in this seminar was PostgreSQL, and all queries were developed and tested using the `psql` terminal client. The schema and test data were created by executing the scripts from Seminar 1 (`create.sql` and `insert.sql`). The Seminar 2 work was implemented as SQL views and queries in a single script (`queries.sql`), which can be executed with `\i` in `psql`.

Correctness was verified by (1) executing each query and checking that the output matches the expected structure from the assignment description, and (2) running smaller sanity checks such as summing `allocated_hours * factor` for a chosen teacher/course instance to confirm totals. For parameterized parts of the script, `psql` variables (`\set`) were used to easily switch course instance, employee, period, and threshold values.

4 Result

4.1 General approach: factored hours and derived hours

A central requirement is to account for preparation time using activity multiplication factors. For both planned and allocated workloads, "factored hours" are calculated as:

$$\text{Factored hours} = \text{Hours} \times \text{Factor}$$

For example, 20 lecture hours with a factor of 3.6 correspond to 72 hours of actual work including preparation.

Administration and examination workload are derived from number of students and HP, and are therefore not stored as columns. Instead, they are computed in queries as:

$$\text{Exam hours} = 32 + 0.725 \cdot \#Students \qquad \text{Admin hours} = 2 \cdot HP + 28 + 0.2 \cdot \#Students$$

4.2 Query 1: planned hours per course instance (current year)

Query 1 returns one row per course instance for the current year. It shows the planned workload per activity after applying multiplication factors, adds derived administration and examination hours, and calculates the final total.

Output (Query 1):

```
course_code|instance_id|hp |study_period|num_students|lecture_hours|tutorial_hours|
-----------+-----------+---+------------+------------+-------------+--------------+
ID2214     |         3 |7.5| P2         |        120 |    108.0000 |      36.0000 |
IV1350     |         4 |7.5| P1         |        180 |           0 |            0 |
IV1351     |         1 |7.5| P2         |        200 |     72.0000 |     192.0000 |
IX1500     |         2 |7.5| P1         |        150 |    158.4000 |            0 |
(4 rows)
```

```
lab_hours | seminar_hours|other_overhead_hours|admin_hours|exam_hours |total_hours
----------+--------------+--------------------+-----------+-----------+-----------
        0 |            0 |            40.0000 |     67.00 |    119.00 | 370.00
  48.0000 |            0 |           100.0000 |     79.00 |    162.50 | 389.50
  96.0000 |     144.0000 |           650.0000 |     83.00 |    177.00 | 1414.00
        0 |     115.2000 |           200.0000 |     73.00 |    140.75 | 687.35
```

4.3 Query 2: allocated hours per teacher for a specific course instance

Query 2 returns one row per teacher involved in a selected course instance, including a breakdown of allocated hours (factored) per activity type, plus a total per teacher. The selected course instance is provided through a `psql` variable (e.g., `:ci`).

Output (Query 2):

```
course_code|instance_id| hp| teacher_name     |designation   |lecture_hours|tutorial_hours
-----------+-----------+---+------------------+--------------+-------------+-------------
IV1351     |         1 |7.5| Adam Assistant   |TA            |           0 |           0
IV1351     |         1 |7.5| Brian Student    |PhD Student   |           0 |           0
IV1351     |         1 |7.5| Leif Linbäck     |Lecturer      |           0 |           0
IV1351     |         1 |7.5| Niharika Gauraha |Lecturer      |           0 |           0
IV1351     |         1 |7.5| Paris Carbone    |Ass. Professor|     72.0000 |           0
(5 rows)
```

Output (Query 2) – part 2 (continued):

```
lab_hours | seminar_hours | other_overhead_hours | admin_hours | exam_hours | total_hours
----------+---------------+----------------------+-------------+------------+-----------
  50.4000 |       18.0000 |                    0 |           0 |          0 |      68.40
  50.4000 |             0 |             100.0000 |           0 |          0 |     150.40
        0 |       72.0000 |             100.0000 |           0 |    62.0000|     234.00
        0 |       72.0000 |             100.0000 |     43.0000 |    61.0000|     276.00
        0 |             0 |             100.0000 |     43.0000 |    61.0000|     276.00
```

4.4 Query 3: allocated hours for a specific teacher (current year)

Query 3 returns one row per course instance for a selected teacher, showing how the teacher's factored allocated hours are distributed across activities and course instances during the current year. The teacher is selected through a `psql` variable (e.g., `:emp`).

Output (Query 3) – part 1:

```
course_code|instance_id | hp|study_period| teacher_name        | lecture_hours|tutorial_hours
-----------+------------+---+------------+---------------------+--------------+--------------+
IV1350     |          4 |7.5| P1         | Niharika Gauraha    |            0 |          0
IX1500     |          2 |7.5| P1         | Niharika Gauraha    |     158.4000 |          0
IV1351     |          1 |7.5| P2         | Niharika Gauraha    |            0 |          0
(3 rows)
```

Output (Query 3) – part 2 (continued):

```
lab_hours | seminar_hours | other_overhead_hours | admin_hours | exam_hours | total_hours
---------+--------------+----------------------+-----------+----------+-----------
      0  |            0 |              40.0000 |         0 |        0 |      40.00
      0  |            0 |             100.0000 |   50.0000 |   30.0000|     338.40
      0  |      72.0000 |             100.0000 |   43.0000 |   61.0000|     276.00
```

4.5 Query 4: teachers allocated to more than N course instances in a period

Query 4 identifies teachers that are allocated to more than a specified number of distinct course instances during a given study period. This is useful for workload monitoring and to detect teachers that are close to or exceeding the business rule limit. The period and threshold are provided through `psql` variables (e.g., `:'period'` and `:n`).
Output (Query 4):

```
employment_id | teacher_name        | study_period | no_of_courses
-------------+---------------------+--------------+------------
500009       | Niharika Gauraha    | P1           |          2
(1 row)
```

5 Discussion

5.1 Query design and readability

The queries follow an OLAP pattern where activity rows are pivoted into separate columns using `SUM(CASE WHEN ... THEN ... END)`. This makes the output easy to read for reporting, since each course instance (or teacher) appears in a single row with a consistent set of columns. Using multiplication factors in the SQL ensures the same logic is applied consistently across reports, and derived administration/examination hours are computed at query-time to avoid storing redundant values.

5.2 Correctness and verification

Correctness was supported by comparing query outputs with expected formats from the assignment description and by executing manual sanity checks for selected cases. For example, summing `allocated_hours * factor` for one teacher in one course instance yields the same total as reported by Query 2. Similar checks were performed across course instances for Query 3 and for distinct course counting in Query 4.

5.3 EXPLAIN ANALYZE and performance

To analyze performance, `EXPLAIN (ANALYZE, BUFFERS)` was executed for Query 4. The output is shown below.

With small test datasets, PostgreSQL often chooses sequential scans because scanning the whole table can be cheaper than using an index when there are few rows. As data grows, indexes become more beneficial. In this schema, indexes on `allocation(employment_id)` and `allocation(course_instance_id)` (and potentially a composite index depending

```
                                                  QUERY PLAN
--------------------------------------------------------------------------------------------------------------------------------------
 Sort  (cost=56.78..56.78 rows=2 width=24) (actual time=1.257..1.258 rows=1.00 loops=1)
   Sort Key: (count(DISTINCT a.course_instance_id)) DESC
   Sort Method: quicksort  Memory: 25kB
   Buffers: shared hit=2
   ->  GroupAggregate  (cost=56.63..56.77 rows=2 width=24) (actual time=0.627..0.629 rows=1.00 loops=1)
         Group Key: a.employment_id
         Filter: (count(DISTINCT a.course_instance_id) > 1)
         Buffers: shared hit=2
         ->  Sort  (cost=56.63..56.65 rows=7 width=20) (actual time=0.462..0.463 rows=5.00 loops=1)
               Sort Key: a.employment_id, a.course_instance_id
               Sort Method: quicksort  Memory: 25kB
               Buffers: shared hit=2
               ->  Hash Join  (cost=28.21..56.53 rows=7 width=20) (actual time=0.443..0.445 rows=5.00 loops=1)
                     Hash Cond: (a.course_instance_id = ci.course_instance_id)
                     Buffers: shared hit=2
                     ->  Seq Scan on allocation a  (cost=0.00..24.50 rows=1450 width=8) (actual time=0.079..0.081 rows=20.00 loops=1)
                           Buffers: shared hit=1
                     ->  Hash  (cost=28.12..28.12 rows=7 width=16) (actual time=0.319..0.320 rows=2.00 loops=1)
                           Buckets: 1024  Batches: 1  Memory Usage: 9kB
                           Buffers: shared hit=1
                           ->  Seq Scan on course_instance ci  (cost=0.00..28.12 rows=7 width=16) (actual time=0.116..0.118 rows=2.00 loops=1)
                                 Filter: (study_period = 'P1'::bpchar)
                                 Rows Removed by Filter: 2
                                 Buffers: shared hit=1
 Planning Time: 3.970 ms
 Execution Time: 2.864 ms
(26 rows)
```

on the query) can help the planner reduce scanned rows and speed up joins and grouping. The EXPLAIN output can be discussed in terms of (1) scan types used, (2) join strategy, (3) estimated versus actual rows, and (4) total execution time and buffers.

6 Comments About the Course

Approximate time spent on Seminar 2 (Task 2): 20 hours (writing queries/views, testing outputs, and performance analysis).