Project Report

Data Storage Paradigms, IV1351

Date

Project members:
[Tianle Niu, 2023110605@stu.hit.edu.cn]

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

This report presents the work for Task 2. The goal of this task was to use the database schema designed in the previous assignment to perform data analysis. While Task 1 focused on structure, this task focuses on inserting data and extracting information for decision-making (OLAP). I have to write 4 queries to perform OLAP.

I worked alone on this task. I was responsible for writing the data generation scripts, designing the SQL queries, and performing the analysis.

2. Literature Study

To establish a theoretical foundation for the implementation of the analytical reports, I studied several key concepts regarding database architecture and SQL performance tuning. The primary sources of information were the course lectures, the course textbook, and the official PostgreSQL documentation.

OLAP vs. OLTP A significant part of this study focused on understanding the distinction between Online Transaction Processing (OLTP) and Online Analytical Processing (OLAP). I learned that while OLTP is designed for managing day-to-day operations with high concurrency and short transactions (e.g., inserting a grade), OLAP is optimized for complex analysis and reporting. This knowledge was applied when designing the queries for this task, where the focus shifted from simple data retrieval to complex aggregations (using `GROUP BY` and `SUM`) to generate cross-tabular reports.

Query Performance Analysis To fulfill the requirement of analyzing query efficiency, I studied the internal working mechanism of the PostgreSQL query planner. I consulted the PostgreSQL documentation regarding the `EXPLAIN` and `EXPLAIN ANALYZE` commands. I studied different scan methods, specifically the difference between **Sequential Scans** and **Index Scans**. I learned that while Index Scans are generally faster for large datasets, Sequential Scans can be more efficient for small tables or when retrieving a large percentage of rows.

2.1. Views and Derived Attributes

Finally, I reviewed the lecture about the concept of database Views and how they can be used to handle derived attributes. This helped in deciding how to handle the "Administration" and "Examination" hours, ensuring that the formulas were consistent and reusable across different reports.

3. Method

3.1. Database Environment and Tools

The DBMS used for this project was PostgreSQL. This system was chosen for its reliability and its support for advanced analysis commands such as `EXPLAIN ANALYZE` (also because I use it in task 1). To write, edit, and execute the SQL scripts, I utilized DBeaver. This tool provided a graphical interface to visualize the table structures and verify the data entry process, while I can conduct queries and observe the result immediately and directly. GitHub was used for version control to manage the SQL scripts for database creation, data insertion, and analysis.

3.2. Work Process

The work process began with reviewing the database schema designed in the previous task to ensure it could support the required analytical reports. While writting Query 1, I identified that the existing dataset was insufficient for meaningful analysis. I identified that the existing dataset, while sufficient for basic structure, lacked specific allocation records required for the new OLAP reports. Specifically, the derived attributes for "Administration" and "Examination" hours were missing from the allocation table. Therefore, I extended the data insertion script to manually assign these calculated hours to specific teachers, ensuring that the workload analysis queries would yield complete and

accurate results. Once the data was prepared, I developed the SQL queries incrementally. I started by creating views for complex calculations, such as the derived attributes for administrative and examination hours, to simplify the main queries. Finally, I selected the most complex allocation query to perform a performance analysis using the database's internal execution planner.

3.3. Verification Strategy

To verify that the SQL queries worked as intended and produced correct data, I used a manual verification method. I did not check every single row in the database; instead, I selected specific test cases for validation. For example, I chose one specific course instance and one specific teacher. I manually calculated the expected total hours and costs using a calculator, strictly following the formulas and multiplication factors provided in the project description. I then compared these manually calculated figures with the results returned by the SQL queries. If the values matched, the query logic was deemed correct. This method ensured that the aggregation logic (SUM, GROUP BY) and the mathematical formulas were implemented correctly in the SQL code.

4. Result

This chapter presents the implementation of the OLAP queries and the analysis of their performance. The solution fulfills all the requirements specified in the project description, including data population, report generation, and efficiency analysis.

4.1. Source Code and Reproducibility

The complete SQL scripts, including database creation, data insertion, and the OLAP queries, are available in the following Git repository:

`https://github.com/turturturturtur/KTHIV1351.git`

The repository contains three distinct scripts to ensure the solution is fully reproducible:

1. `001_CreateDataBase.sql`: Creates the database tables and relationships.
2. `002_InsertData.sql`: Populates the database with course instances, employees, and specific allocation records required for the analysis.
3. `003_Query.sql`: Contains the four analytical queries and the performance analysis command.

The results presented below can be reproduced by executing these scripts in the order listed above.

4.2. OLAP Queries and Reports

4.2.1. Query 1

Requirement: Calculate total hours for course instances, including derived attributes for administrative and examination duties.
Explanation: This query generates a comprehensive plan for course instances in the current year. It aggregates the planned hours for standard activities (Lectures, Labs, etc.) and multiplies them by their respective factors. Furthermore, it dynamically calculates the "Admin" and "Exam" hours using the specified formulas.

$$\text{Admin Hours} = 2 \times \text{HP} + 28 + 0.2 \times \#\text{Students} \tag{1}$$

$$\text{Exam Hours} = 32 + 0.725 \times \#\text{Students} \tag{2}$$

Output: Figure 1 shows the output where the derived attributes (Admin/Exam) are correctly calculated based on the student count and HP for each course.

	Course Code	Instance ID	HP	Period	# Students	Lecture	Tutorial	Lab	Seminar	Other	Admin	Exam	Total Hours
1	DD1338	2025-70112	7.5	P1	220	129.6	57.6	43.2	0	220	87	191.5	728.9
2	IX1500	2025-50413	7.5	P1	150	158.4	0	0	115.2	200	73	140.8	687.4
3	ID2214	2025-50341	7.5	P2	120	100.8	72	0	36	120	67	119	514.8
4	IV1351	2025-50273	7.5	P2	200	72	192	96	144	650	83	177	1,414
5	EQ2010	2025-70113	7.5	P3	160	108	43.2	48	0	140	75	148	562.2
6	IV1350	2025-60104	7.5	P3	180	115.2	0	86.4	32.4	180	79	162.5	655.5
7	ID2202	2025-70111	7.5	P4	140	144	48	48	0	160	71	133.5	604.5
8	SF1688	2025-70114	7.5	P4	110	93.6	0	0	54	100	65	111.8	424.4

Figure 1: Output of Query 1: Planned hours with derived attributes.

4.2.2. Query 2

Requirement: Calculate the actual allocated hours for a specific course instance, broken down by teacher and activity.
Explanation: This query focuses on the course instance `IV1351` (ID: 2025-50273). It joins the `allocation` table with the `employee` table to show exactly how the workload is distributed among teachers. It applies the multiplication factor to every activity to reflect the true working hours.
Output: As shown in Figure 2, the report lists each teacher involved in the course and their total hours. Note that the "Admin" and "Exam" hours previously calculated in Query 1 are now assigned to specific teachers (e.g., Paris Carbone).

	Course Code	Instance ID	HP	Teacher	Designation	Lecture	Tutorial	Lab	Seminar	Other	Admin	Exam	Total
1	IV1351	2025-50273	7.5	Paris Carbone	Ass. Professor	72	0	0	0	100	83	177	432
2	IV1351	2025-50273	7.5	Leif Linbäck	Lecturer	0	0	0	72	0	0	0	72
3	IV1351	2025-50273	7.5	Niharika Gauraha	Lecturer	0	0	0	72	0	0	0	72
4	IV1351	2025-50273	7.5	Adam	TA	0	0	48	0	0	0	0	48
5	IV1351	2025-50273	7.5	Brian	PhD Student	0	0	48	0	0	0	0	48

Figure 2: Output of Query 2: Allocation breakdown for course IV1351.

4.2.3. Query 3

Requirement: Calculate the total allocated hours for a specific teacher across all courses in the current year.
Explanation: This query aggregates the workload for the teacher "Paris Carbone". It sums up all activities from different course instances (P2, P3, P4) to provide a total workload overview. This verifies that the system can track an employee's load across the entire academic year.
Output: Figure 3 illustrates Paris Carbone's workload distribution across different periods and activities.

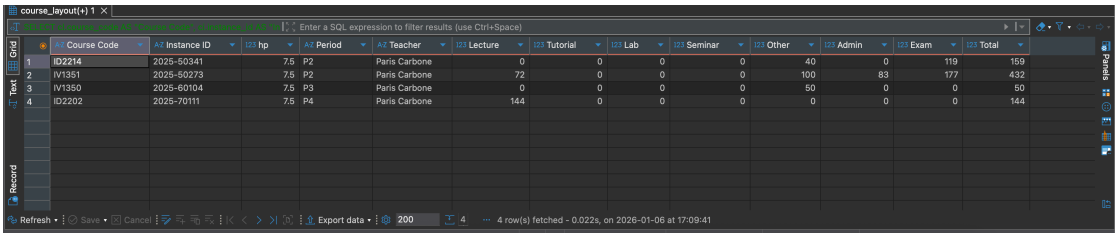

Figure 3: Output of Query 3: Workload report for Paris Carbone.

4.2.4. Query 4

Requirement: List teachers allocated to more than a specific number of course instances in a period.
Explanation: This query identifies potential workload issues. It groups allocations by teacher and period, counting the number of distinct course instances. A `HAVING COUNT(...) > 1` rule is applied to filter out teachers who are managing more than one course in a single period.
Output: Figure 4 shows the teachers who exceed the specified course limit in period P2.

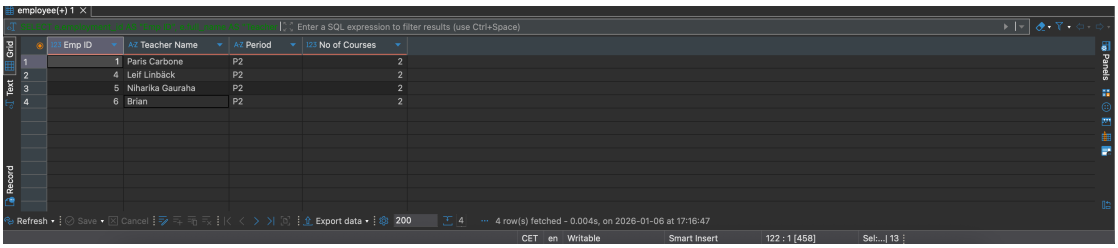

Figure 4: Output of Query 4: Teachers with multiple courses in one period.

4.3. Performance Analysis

Requirement Analyze the efficiency of one query using `EXPLAIN ANALYZE`.

Explanation: I executed the `EXPLAIN ANALYZE` command on Query 2 (Actual Allocation) to evaluate its execution plan. This query was chosen because it involves multiple joins (4 tables) and aggregation, making it a good candidate for analysis.
Output: The execution plan is shown in Figure 5. The analysis of this plan, including the scan methods and execution time, is discussed in detail in the Discussion chapter.

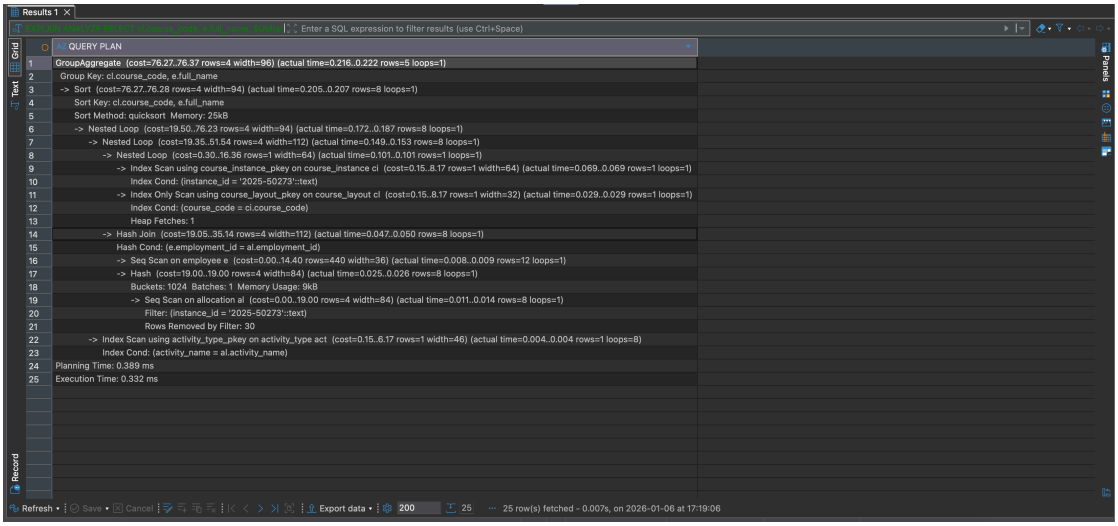

Figure 5: Output of EXPLAIN ANALYZE for Query 2.

5. Discussion

In this chapter, I evaluate my solution based on the assessment criteria provided for the course. The evaluation focuses on query performance, the use of database views, and the impact of design choices on query simplicity.

5.1. Performance Analysis with EXPLAIN ANALYZE

As required by the mandatory assessment criteria, I analyzed the query plan for Query 2 (Actual Allocated Hours) using the `EXPLAIN ANALYZE` command (I conducted the experiment using PostgreSQL).

Execution Time and Bottlenecks: The analysis showed a total execution time of approximately **0.332 ms**, with a planning time of 0.389 ms. The DBMS spent the most time performing a **Sequential Scan (Seq Scan)** on the `allocation` table.

Is this reasonable? Yes, this behavior is entirely reasonable given the current volume of data. The `allocation` table contains only a small number of rows (less than 100 in the test set). The PostgreSQL optimizer correctly determined that scanning the entire table sequentially is faster than the overhead associated with loading and traversing an index (Random I/O).

Optimization Potential: While the current plan is optimal for the test data, the Sequential Scan could become a performance bottleneck if the university's data grows to millions of records. In a production environment with a large dataset, adding a specific index on `allocation.instance_id` would be necessary. This would force the planner to switch from a Sequential Scan to an Index Scan, ensuring the query remains scalable.

5.2. Usage of Views

One of the assessment criteria asks whether views are used to improve understanding. I implemented a view named `view_course_planned_hours` for the first query.

- **Reasoning:** The calculation for "Administration" and "Examination" hours involves derived attributes based on specific formulas (e.g., Admin $= 2 \times \text{HP} + \dots$).
- **Benefit:** By storing this logic in a view, I avoided repeating complex mathematical formulas in the main OLAP queries. This ensures that if the university changes the formula in the future, it only needs to be updated in one place (the View definition) rather than in every report query. This significantly improves the maintainability and readability of the SQL code.

5.3. Database Design and Simplification

Regarding the criteria on database design changes, I adopted a strategy to **unify the specification** of all teaching activities.

- **Change:** I treated the derived attributes "Administration" and "Examination" as standard activities by inserting them into the `activity_type` table with a multiplication factor of **1.0**.
- **Evaluation:** Since Admin and Exam are calculated by formulas (derived attributes), they typically do not have a "preparation factor" like Lectures do. However, by explicitly defining them with a factor of 1, I standardized the data structure.
- **Impact:** This allowed Query 2 to use a single, generic calculation logic: `SUM(hours * factor)`. Without this design, the query would require separate logic branches to handle "Activities with factors" versus "Derived Attributes". This change significantly simplified the SQL code and made the system consistent, as every activity is processed in exactly the same way.

5.4. Query Structure

Finally, I ensured that no correlated subqueries were used in the solution. Correlated subqueries can lead to poor performance ($O(N^2)$ complexity) because the inner query is executed for every row of the outer query. Instead, I utilized standard `JOIN` operations and `GROUP BY` aggregations, which allow the database to use efficient algorithms like Hash Joins, as confirmed by the query plan analysis.

6. Comments About the Course

In total, I spent approximately **9 hours** completing this assignment. The time distribution was roughly as follows:

- **Preparation (3 hours):** This included reviewing the relevant lecture notes and studying the official documentation to understand the usage and output of the `EXPLAIN ANALYZE` command.
- **Implementation (3 hours):** This involved writing the SQL scripts, populating the database with test data, and debugging the queries (some of time was spent on thinking about how to deal with allocation table).
- **Report Writing (3 hours):** This time was spent documenting the solution, analyzing the query plans, and formatting the report.

Feedback: I found this assignment very helpful for bridging the gap between theory and practice. Specifically, using `EXPLAIN ANALYZE` gave me a concrete understanding of how the database optimizer works, which was a concept I previously only understood theoretically. The workload was reasonable and the instructions were clear.

A. Original Design Notes

This appendix includes the scanned copies of my original handwritten notes and sketches created during the brainstorming and initial design phases of this project. In this note, I think about how to deal with the allocation tabel and eventually changed the schema of the relationship allocation.
Note on Language: Please be advised that these notes are written in Chinese. As a non-native English speaker, I chose to use my native language during the conceptualization stage to minimize cognitive load and ensure an uninterrupted flow of logic and creativity.

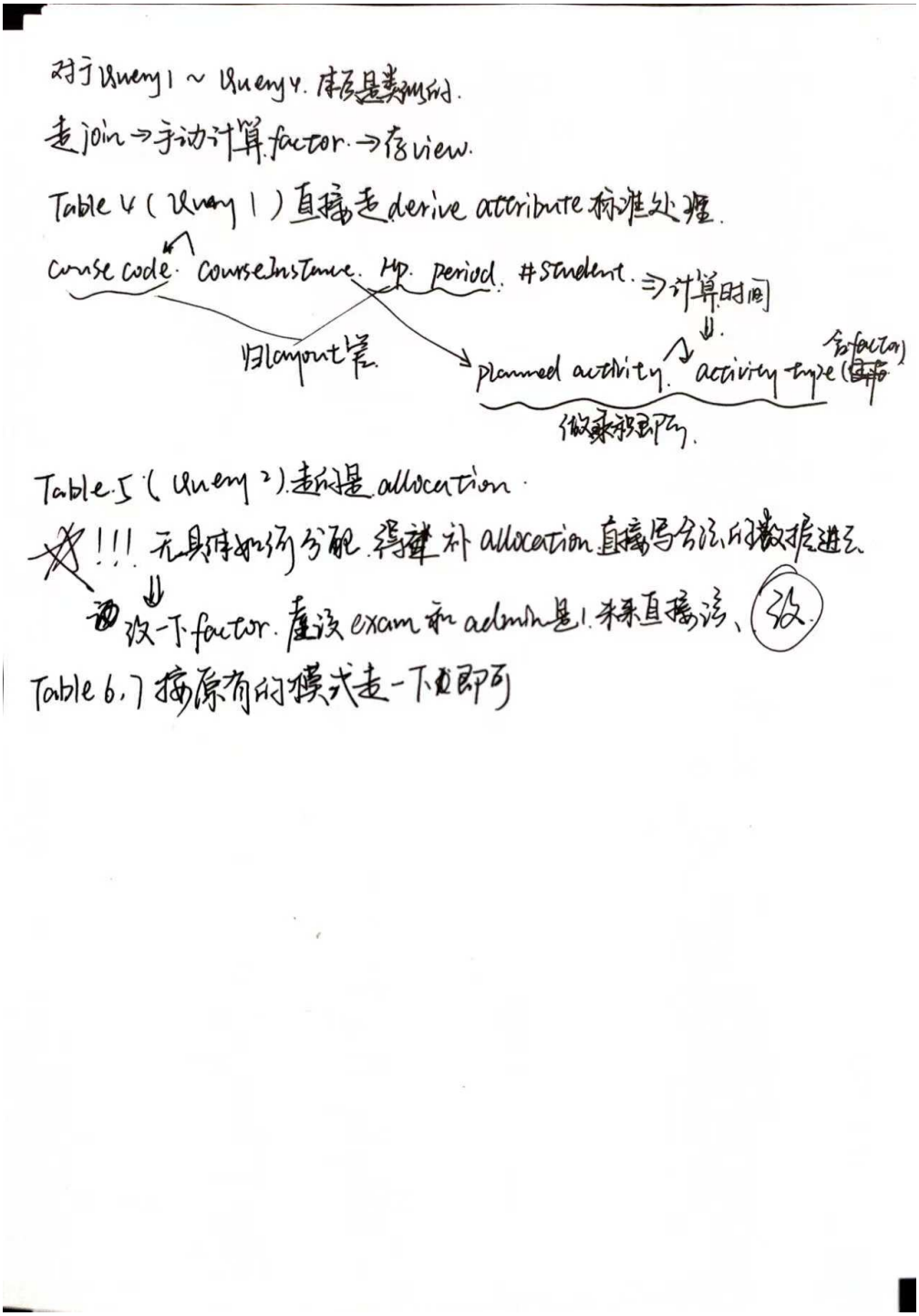

Figure 6: Handwritten notes: Derivation of formulas and table relationships.