Project Report

Data Storage Paradigms, IV1351

Date

Project members:
[Elias Tosteberg, eliasto@kth.se]

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

For this seminar the task is to design OLAP queries. These should run on a database Designed for a university. These queries are:

- **1. Planned hours calculations:**

- **2. Calculate actual allocated hours for a course.**

- **3. Calculate the total allocated hours (with multiplication factors) for a teacher, only for the current years' course instances.**

- **4. List employee ids and names of all teachers who are allocated in more than a specific number of course instances during the current period.**

2 Literature Study

When researching this project information was gathered from the SQL lecture, the SQL chapter of the book and the tips and tricks document. The SQL lecture and coursebook gave good information on how to use the SQL language. There it was explained how to write the queries. The tips and tricks document for example gave good information on subqueries and to avoid correlated subqueries due to them having to execute for every row of the table causing bad performance.

3 Method

The project was made in PostgresSQL written in VSCode. When making a query the first step was to figure out what data was needed and join the tables so that all the needed data was accessible. Second step was to make the easy columns in the select statement. Third step was to figure out what aggregate functions as well as grouping the rows so that the data that isn't conveniently stored can be read. While developing the queries were tested by comparing the output with the result that can be calculated manualy.

4 Result

The queries are stored as Query.sql in github.com/Sailet03/Seminar-IV1351-2025.

4.1 Number one: Planned hours calculations

The query will calculate a breakdown of all the hours needed for every course instance for a given year. First the query will join the CourseLayout, Instance, PlannedActivities and ActivityType to get all the required information. Since the hours are stored with the only difference being the type we need to do some math to separate the different activity types into separate columns. This is done by using SUM together with WHERE to sum together all activities of the same type into a value that can be showed in a column. The GROUP BY statement ensures that everything belonging to the same instance is treated together and the WHERE ensures that only the specified year is shown. Here is the output of the query.

Course Code	Course Instance ID	HP	Period	# Students	Lecture Hours	Tutorial Hours	Lab Hours	Seminar Hours	Other Overhead Hours	Admin	Exam	Total Hours
CS101	2024-1	7	P1	30	137	34	53	24	9	88	108	453
MA201	2024-2	5	P2	25	169	42	70	30	11	80	100	502
PH110	2024-3	10	P3	40	137	34	55	24	9	100	122	481

4.2 Number two: Calculate actual allocated hours for a course

This query will break down how much time is allocated to per employee for a specific course instance. This query works mostly the same since most of the columns are the same. Since it required more information than number one it also had to JOIN the

EmployeesPlannedActivities, Employees, Titles and Person. The WHERE statement ensures that the correct course instance is selected. The GROUP By statement is ensuring that the employees are grouped together.

Course Code	Course Instance ID	HP	Teacher's Name	Designation	Lecture Hours	Tutorial Hours	Lab Hours	Seminar Hours	Other Overhead Hours	Admin	Exam	Total Hours
CS101	2024-1	7	Alice Johnson	Assistant Professor	0	0	24	0	5	0	0	29
CS101	2024-1	7	Bob Smith	Associate Professor	0	0	29	0	4	0	0	33
CS101	2024-1	7	Frank Miller	Lecturer	72	0	0	11	0	0	54	137
CS101	2024-1	7	Grace Lee	Senior Lecturer	65	0	0	13	0	0	54	132
CS101	2024-1	7	Hannah Kim	Lecturer	0	16	0	0	0	44	0	60
CS101	2024-1	7	Ian Clark	Lab Assistant	0	18	0	0	0	44	0	62

4.3 Number three: Calculate the total allocated hours for a teacher, only for the current years' course instances

This query is also similar to the first 2. The query gets the hours that are assigned to a specific employee for all course instances in a certain year. Here The WHERE statement filters on year and name of an employee. The rows are grouped by the teacher name and year.

Course Code	Course Instance ID	HP	Teacher's Name	Lecture Hours	Tutorial Hours	Lab Hours	Seminar Hours	Other Overhead Hours	Admin	Exam	Total Hours
CS101	— 2024-1	7	Frank Miller	72	0	0	11	0	0	54	137
MA201	— 2024-2	5	Frank Miller	0	20	0	0	0	40	0	60
PH110	— 2024-3	10	Frank Miller	0	0	29	0	4	0	0	33

4.4 Number four: List employee ids and names of all teachers who are allocated in more than a specific number of course instances during the current period

The final Query shows all the employees that have scheduled activities belonging to more than a certain number of instances in a specific period. The query works by joining Employees, Person, EmployeesPlannedActivities, PlannedActivities and Instance together. The Query then uses COUNT DISTINCT to count the number of instances in the current period. The HAVING filter is then used to ensure only the teachers with more than the lower amount is shown.

Employment ID	Teacher's Name	Period	No of Courses
3	Alice Johnson	P1	1
4	Bob Smith	P1	1
1	Frank Miller	P1	1
2	Grace Lee	P1	1
5	Hannah Kim	P1	1
6	Ian Clark	P1	1

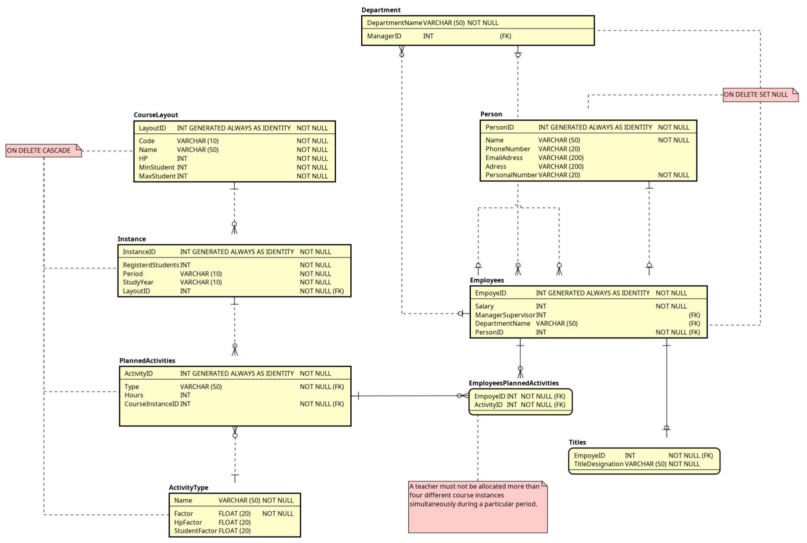
Department
DepartmentName VARCHAR (50) NOT NULL
ManagerID INT (FK)
ON DELETE SET NULL
CourseLayout
LayoutID INT GENERATED ALWAYS AS IDENTITY NOT NULL
Code VARCHAR (10) NOT NULL
Name VARCHAR (50) NOT NULL
HP INT NOT NULL
MinStudent INT NOT NULL
MaxStudent INT NOT NULL
ON DELETE CASCADE
Person
PersonID INT GENERATED ALWAYS AS IDENTITY NOT NULL
Name VARCHAR (50) NOT NULL
PhoneNumber VARCHAR (20)
EmailAdress VARCHAR (200)
Adress VARCHAR (200)
PersonalNumber VARCHAR (20) NOT NULL
Instance
InstanceID INT GENERATED ALWAYS AS IDENTITY NOT NULL
RegisterdStudents INT NOT NULL
Period VARCHAR (10) NOT NULL
StudyYear VARCHAR (10) NOT NULL
LayoutID INT NOT NULL (FK)
Employees
EmpoyeID INT GENERATED ALWAYS AS IDENTITY NOT NULL
Salary INT NOT NULL
ManagerSupervisor INT (FK)
DepartmentName VARCHAR (50) (FK)
PersonID INT NOT NULL (FK)
PlannedActivities
ActivityID INT GENERATED ALWAYS AS IDENTITY NOT NULL
Type VARCHAR (50) NOT NULL (FK)
Hours INT
CourseInstanceID INT NOT NULL (FK)
EmployeesPlannedActivities
EmpoyeID INT NOT NULL (FK)
ActivityID INT NOT NULL (FK)
Titles
EmpoyeID INT NOT NULL (FK)
TitleDesignation VARCHAR (50) NOT NULL
ActivityType
Name VARCHAR (50) NOT NULL
Factor FLOAT (20) NOT NULL
HpFactor FLOAT (20)
StudentFactor FLOAT (20)
A teacher must not be allocated more than four different course instances simultaneously during a particular period.

Figure 1: Database schema

5 Discussion

When creating the queries the database were not changed to make the query simpler. It would have been possible to store the employee titles in the employee table instead of in their own table, but it was not a big difference in simplicity. It would have also made more work since that would require the data script to be modified. The queries themselves are not very complicated, but the aggregate functions in query 1,2 and 3 are quite complicated since they are filled with math. Thsi does however not seem to cause any significant performance issues. Query 1 was analyzed with EXPLAIN. Most of the computation seem to be the multiple hash joins that is used to collect all the information. None of the Sequential Scans seems to be causing exponential complexity and the query lacks any subqueries so Correlated Subqueries are not a concern.

```
                                                            QUERY PLAN
----------------------------------------------------------------------------------------------------------

 Sort  (cost=10.28..10.30 rows=9 width=122)
   Sort Key: c.code
   ->  HashAggregate  (cost=9.82..10.13 rows=9 width=122)
         Group Key: c.code, i.instanceid, c.hp
         ->  Hash Join  (cost=3.31..5.62 rows=42 width=50)
               Hash Cond: ((pa.type)::text = (a.name)::text)
               ->  Hash Join  (cost=2.16..4.28 rows=42 width=38)
                     Hash Cond: (i.layoutid = c.layoutid)
                     ->  Hash Join  (cost=1.09..2.95 rows=42 width
                        =32)
                           Hash Cond: (pa.courseinstanceid = i.
                              instanceid)
                           ->  Seq Scan on plannedactivities pa  (
                              cost=0.00..1.56 rows=56 width=16)
                           ->  Hash  (cost=1.05..1.05 rows=3 width
                              =20)
                                 ->  Seq Scan on instance i  (cost
                                    =0.00..1.05 rows=3 width=20)
                                       Filter: ((studyyear)::text
                                          = '2024'::text)
                     ->  Hash  (cost=1.03..1.03 rows=3 width=14)
                           ->  Seq Scan on courselayout c  (cost
                              =0.00..1.03 rows=3 width=14)
               ->  Hash  (cost=1.07..1.07 rows=7 width=20)
                     ->  Seq Scan on activitytype a  (cost
                        =0.00..1.07 rows=7 width=20)
```