

SWitCH Dev 2024/2025

Database Project

Portal AI (PAI)

Group 2

Ana Catarina Magalhães Conde (1241897)

Cláudio Alexandre Ferraz Cunha (1241901)

Fernando Júlio Araújo Moreira (1241905)

Inês Isabel Leal Barbosa (1241909)

João Pedro Gomes de Lemos Pinheiro (1241913)

Mélanie Audrey Gomes Barbosa (1241917)

Pedro Miguel Alves Soares (1241921)

Sérgio Fernando Pinto Ramos (1241925)

1. Relational Data Model

1.1. Programmes and Courses

In the context of programmes and courses, we define the entity `student_programme` to include an attribute named `status`, which can only take two values: "active" or "alumni". If the student is an alumni, the completion certificate will have the value "Y". If the student is still active, this attribute will have the value "N". The completion certificate includes the final grade for every course, which can be obtained from the `student_course` table through the `final_grade` attribute.

The sentence - "A programme may change over time, but there can be only one valid study plan of the programme at a time" - implies that a programme can have multiple study plans but only one active at a specific time. Therefore, we can determine which one is currently valid by using the attribute `school_year`.

As stated in the project description, the equivalence map describes the relationship between courses in the new study plan and those in the previous study plan. Therefore, our entity equivalence map includes two relationships with the course entity (one referencing the course in the new study plan and the other referencing the course in the previous study plan) and two relationships with the study plan entity.

We added an `attempted_improvement` attribute to the table `student_course` to symbolize the fact that the student can only attempt grade improvement once. For this, the `attempted_improvement` attribute will have only two possible values: N (No) and Y (Yes), indicating whether that student has not attempted or has already attempted grade improvement for that specific course. The value of `attempted_improvement` can only be updated to Y if the student is already approved in that course upon re-enrollment (we can confirm their approval status by referencing the `final_grade` attribute). We specified that `attempted_improvement` can only be performed once by students who were already previously approved to that specific course. Therefore, if the `final_grade` is below 10 (was not approved) or NULL (is currently enrolled in the course), the `attempted_improvement` is N by default.

For the `student_course` table, we applied a trigger to replace the original `final_grade` with the grade obtained by the student following improvement attempt, if this grade is superior to the original `final_grade`. This way, we do not need to create an additional attribute to store the improved grade value.

If a student's final grade in a specific course is below 10.00, it means that the student was not approved for the course. However, if the final grade is `NULL`, it indicates that the student is still taking the course and is yet to receive a final grade.

When analyzing the query number 10, we identified the necessity to include a "prerequisites" attribute in the course table. This attribute will indicate whether a course has prerequisites, with the values "yes" for courses that do, and "no" for those that do not.

We assigned the attribute `semester` to the `study_plan_course` entity. This ensures that a `study_plan_course` is confined to a single semester and does not span across multiple semesters.

We added a `year_in_programme` to the `course_edition` to specify the year (within the programme) in which a specific `course_edition` was lectured. The `year_in_programme` can therefore only have the values from 1 to 5, with 5 corresponding to the integrated master's degree (the longest type of programme). This was created as a response to query 13, in order to access the courses taught in the 3rd curricular year.

1.2. Students

Regarding students, we can calculate the total number of ECTS a student is enrolled in by accessing the `student_course` entity, which is related to the `course` entity containing the attribute `number_ects`.

We identified the need for an attribute `enrollment_date_time` in the entity `student_course` when implementing query number 11, which requests the date and time of students' enrollment.

1.3. Teachers

A teacher can teach multiple course editions, and each course edition can have multiple teachers. We chose not to connect teacher directly to course because teachers may vary between courses across different school years. Instead, this relationship was implemented through the `course_edition_teacher` table.

1.4. Classrooms

We assigned the term "class" to refer to a group of students enrolled in a specific course or attending lessons together. The term "course_class" represents a single lesson conducted in a classroom. Therefore, when it is stated that "Each class has a set of programme classes assigned to it. At least one," we assumed that a class is associated with at least one `course_class` and can have multiple, and that each `course_class` is exclusively linked to a single class.

We established a relationship between class and classroom, where each class is assigned to a single classroom, while a classroom can accommodate multiple classes, at different times. This aligns with the project description, which specifies that a classroom may be linked to a programme class without being tied to a specific course.

Each classroom will have a `classroom_type` (lecture room, a plain classroom or a lab).

We also find the need to classify the `course_class_type` (PL, OT, T, TP). Since it is necessary to access the study plan and view the number of weekly hours for each type of class, we connected `course_class_type` to `study_plan_course` (a N-N relationship), which led to the creation of `study_plan_course_class_type`.

1.5. Timetable

We assume that each student has multiple weekly timetables and since it is a N-N relationship, we created the `student_weekly_timetable` entity. Within a specific `start_date` and `end_date`, this timetable includes multiple course classes. However, each `course_class` - uniquely identified by its distinct ID - can belong to only one weekly timetable.

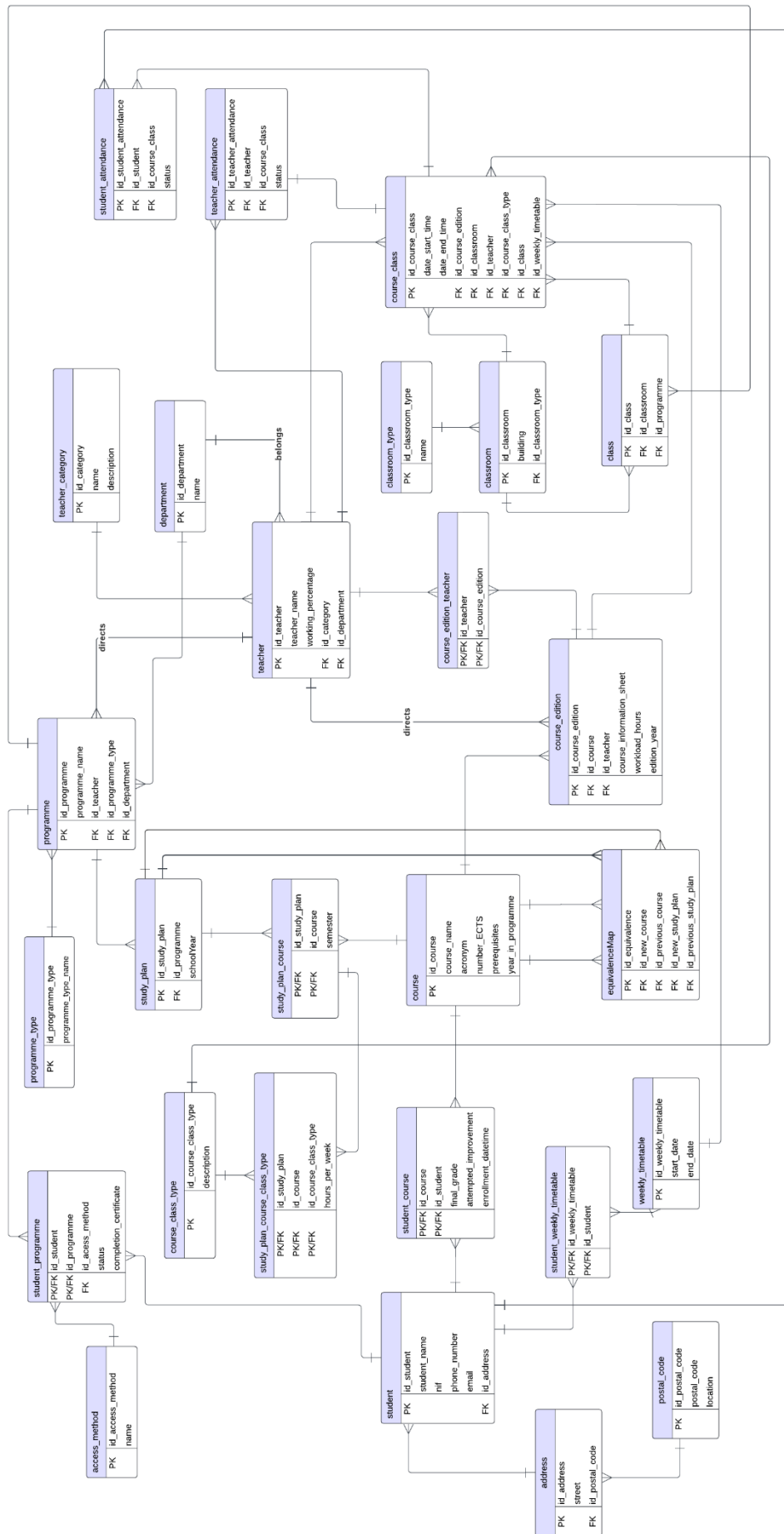
1.6. Attendance

To track the attendance of teachers and students in a specific `course_class`, we created two entities: `teacher_attendance` and `student_attendance`. Both entities include a status attribute, indicating whether the individual was present, absent or late (since it is also important to take into account punctuality rules, as mentioned in the last paragraph).

Integrity Constraints:

1. The attribute status in `Student_programme` will only assume the values “alumni” or “active”.
2. If `id_completion_certificate` has the value “N”, the student has not graduated and is still active. If it has the value “Y”, the student is considered an alumni.
3. A course has a name and an acronym, which are unique in the programme, but not across programmes
4. Grades in a course must be in the [0, 20] range.
5. A student that has 10 or more in the final grade of a course is approved on that course.
6. A student cannot be enrolled in a course it has approved before, unless it is for grade improvement, and can only do this once.
7. A student can only attempt grade improvement if he has previously been approved for that specific course.

8. If a student enrolls for grade improvement and achieves a grade lower than their original one, the original grade will be maintained.
9. There are limits for the minimum and maximum number of credits a student may be enrolled in a school year.
10. The attribute name of the building is restricted to a single letter ranging from A to H.
11. The attribute start_time of the entity course_class can only be 10 minutes after the hour or 40 minutes after the hour.
12. A course_class must have the duration of at least 1 hour.
13. A course_class is restricted to occur between 08:10 and 23:40.
14. Each class has a set of programme classes assigned to it. At least one.
15. Each course_class may not encompass multiple days.
16. A course cannot encompass multiple semesters.
17. The email must follow a valid format.



Queries

Query 1

The objective of this query is to identify the students with the highest number of subjects in arrears compared to all other students in the "Switch" programme. The query is designed to find students in the "Switch" programme who have the most subjects in arrears (subjects with a final grade below 10). We assume that if a student received a final grade below 10.00 in a course, they were not approved for that course.

The query works by:

1. Joining relevant tables (student, student_programme, programme, and student_course) to link students with their courses and programmes;
2. Filtering for students in the "Switch" program who have at least one course with a grade below 10.00;
3. Using a subquery to calculate the number of subjects in arrears for each student;
4. Using the HAVING clause to only return students with the maximum number of subjects in arrears.

```
SELECT student.student_name
FROM student_programme
JOIN programme ON student_programme.id_programme =
programme.id_programme
JOIN student_course ON student_programme.id_student =
student_course.id_student
JOIN student ON student_programme.id_student = student.id_student
```



```

WHERE programme.programme_name = 'Switch' AND student_course.final_grade <
10.00
GROUP BY student.student_name
HAVING COUNT(student_course.id_course) = (SELECT MAX(subjects_in_arrears)
FROM ( SELECT COUNT(student_course.id_course) AS subjects_in_arrears
FROM student_programme
JOIN programme ON student_programme.id_programme =
programme.id_programme
JOIN student_course ON student_programme.id_student =
student_course.id_student
WHERE programme.programme_name = 'Switch' AND student_course.final_grade <
10.00
GROUP BY student_programme.id_student
)
);

```

	STUDENT_NAME
1	Luis Sousa
2	Sofia Barros

Query 2

The objective of this query is to identify students who have more total enrollments than the student with the highest number of enrollments among those enrolled in at least one "Civil Engineering" course. The query first identifies the student with the maximum number of enrollments among students associated with "Civil Engineering". Then, it compares this maximum value with the total enrollments of all students to find those who exceed this benchmark.

The query works by:

1. Identify all students enrolled in "Civil Engineering" (civil_students).
2. Calculate the total enrollments for those students (civil_students_total).

3. Determine the highest enrollment among them (max_civil).
4. Compute the total enrollments for all students (all_students).
5. Compare each student's total enrollments with the maximum enrollment from max_civil and return those with higher totals.

```
WITH civil_students AS (  
    SELECT DISTINCT student.id_student, student.student_name  
    FROM student  
    JOIN student_course ON student.id_student = student_course.id_student  
    JOIN course ON student_course.id_course = course.id_course  
    WHERE course.course_name = 'Civil Engineering'  
)  
,  
civil_students_total AS (  
    SELECT sc.id_student, COUNT(sc.id_course) AS total_matriculas  
    FROM student_course sc  
    WHERE sc.id_student IN (SELECT id_student FROM civil_students)  
    GROUP BY sc.id_student  
)  
,  
max_civil AS (  
    SELECT MAX(total_matriculas) AS max_matriculas  
    FROM civil_students_total  
)  
,  
all_students AS (  
    SELECT student.id_student, student.student_name,  
    COUNT(student_course.id_course) AS total_matriculas  
    FROM student  
    JOIN student_course ON student.id_student = student_course.id_student  
    GROUP BY student.id_student, student.student_name  
)  
SELECT s.id_student, s.student_name, s.total_matriculas  
FROM all_students s  
WHERE s.total_matriculas > (SELECT max_matriculas FROM max_civil)
```

ORDER BY s.total_matriculas DESC;

	ID_STUDENT	STUDENT_NAME	TOTAL_MATRICULAS
1	34	Ana Oliveira	5
2	30	Nuno Cunha	5
3	31	João Silva	5
4	32	Maria Sousa	5
5	33	Pedro Costa	5
6	35	Lucas Pereira	5
7	25	Mariana Ferreira	5
8	20	Fernando Lopes	3
9	1	Filipa Martins	3

Query 3

The objective of this query is to identify students who are enrolled in courses with a workload of less than 60 hours, along with their associated teachers. The query helps to analyze which students are taking courses that have a relatively lighter workload.

The query works by:

1. Joining the relevant tables (student, student_course, course_edition, and teacher) to link students with their courses and teachers;
2. Filtering for courses that have a workload of less than 60 hours;
3. Selecting the student name and teacher name for each course that meets the workload condition.

SELECT

s.student_name AS Student,

t.teacher_name AS Teacher

FROM student s

JOIN student_course sc ON s.id_student = sc.id_student

JOIN course_edition ce ON sc.id_course = ce.id_course

JOIN teacher t ON ce.id_teacher = t.id_teacher

WHERE ce.workload_hours < 60;

	STUDENT	TEACHER
1	Filipa Martins	Carlos Félix
2	Filipa Martins	Tiago Andrade
3	Sara Gomes	Sónia Pizarro
4	Carla Mendes	Nuno Silva
5	Luis Sousa	Bruno Oliveira
6	Antonio Oliveira	Olga Castro
7	Fernando Lopes	Isabel Pereira
8	Fernando Lopes	Tiago Andrade
9	Catarina Neves	Tiago Andrade
10	Pedro Teixeira	Joaquim Alves

	STUDENT	TEACHER
11	Mariana Ferreira	Nuno Silva
12	Rafael Santos	Isabel Figueiredo
13	Carlos Rocha	Carla Lopes
14	Nuno Cunha	Olga Castro
15	Nuno Cunha	Tiago Andrade
16	João Silva	Luís Nogueira
17	João Silva	Alexandre Gouveia
18	João Silva	Nuno Silva
19	Maria Sousa	Luís Nogueira
20	Maria Sousa	Alexandre Gouveia

	STUDENT	TEACHER
21	Maria Sousa	Nuno Silva
22	Pedro Costa	Luís Nogueira
23	Pedro Costa	Alexandre Gouveia
24	Pedro Costa	Nuno Silva
25	Ana Oliveira	Luís Nogueira
26	Ana Oliveira	Alexandre Gouveia
27	Ana Oliveira	Nuno Silva
28	Lucas Pereira	Luís Nogueira
29	Lucas Pereira	Alexandre Gouveia
30	Lucas Pereira	Nuno Silva

Query 4

List the number, name, and course average of the top 20% of students and the bottom 20%. The average should be calculated considering only students who have passed all courses in their programme. The best students should be marked with “Top 20%” and the worst with “Bottom “20%”:

1. Filters students who passed all courses (grades ≥ 10) and calculates their average grade.
2. Ranks students by average grade in descending order for top 20% and ascending order for bottom 20%.
3. Combines top 20% and bottom 20% students with labels ("Top 20%" or "Bottom 20%").

```

WITH students_approved AS (
  SELECT
    student.id_student,
    student.student_name,
    AVG(student_course.final_grade) AS average_grade
  FROM
    student student
  JOIN
    student_course student_course ON student.id_student=
    student_course.id_student
  WHERE

```

```

        student_course.final_grade >= 10
    GROUP BY
        student.id_student, student.student_name
),
student_count AS (
    SELECT COUNT(*) AS total_students FROM students_approved
),
ranked_students AS (
    SELECT
        id_student,
        student_name,
        average_grade,
        NTILE(5) OVER (ORDER BY average_grade DESC) AS percentile_rank
    FROM
        students_approved
)
SELECT
    id_student,
    student_name,
    average_grade,
    CASE
        WHEN percentile_rank = 1 THEN 'Bottom 20%'
        WHEN percentile_rank = 5 THEN 'Top 20%'
    END AS classification
FROM
    Ranked_students, student_count
WHERE
    percentile_rank IN (1, 5)
ORDER BY
    classification, average_grade DESC;

```

	ID_STUDENT	STUDENT_NAME	AVERAGE_GRADE	CLASSIFICATION
1	3	Carlos Santos	20	Bottom 20%
2	21	Catarina Neves	20	Bottom 20%
3	30	Nuno Cunha	20	Bottom 20%
4	13	Carla Mendes	19.5	Bottom 20%
5	15	Patricia Dias	11	Top 20%
6	9	André Almeida	10	Top 20%
7	23	Isabel Silva	10	Top 20%

Query 5

In this query, we have to list the number and names of students of Professor Ângelo Martins, that took his classes in 2021 and 2023 and also who had classes with Professor Rosa Reis.

This query will select the names of the students and count them from the table students. After that, it will verify which students are on the list of students who attended courses taught by teachers Ângelo Martins and Rosa Reis.

```

SELECT s.student_name, COUNT(s.id_student) AS num_students
FROM student s
WHERE s.id_student IN (
    SELECT DISTINCT sc.id_student
    FROM student_course sc
    JOIN course_class cc1 ON sc.id_course = cc1.id_course_edition
    WHERE cc1.id_teacher = (SELECT id_teacher FROM teacher WHERE
teacher_name = 'Ângelo Martins')
    AND (EXTRACT(YEAR FROM cc1.date_start_time) = 2021 OR EXTRACT(YEAR
FROM cc1.date_start_time) = 2023)
    AND sc.id_student IN (
        SELECT sc2.id_student
        FROM student_course sc2
        JOIN course_class cc2 ON sc2.id_course = cc2.id_course_edition

```

```

WHERE cc2.id_teacher = (SELECT id_teacher FROM teacher WHERE
teacher_name = 'Rosa Reis')
)
)
GROUP BY s.student_name;

```

	STUDENT_NAME	NUM_STUDENTS
1	Mariana Ferreira	1
2	Nuno Cunha	1

Query 6

The goal is to list teachers who only taught courses with workloads between 61 and 79 hours.

```

SELECT
    t.teacher_name
FROM
    teacher t
JOIN
    course_edition_teacher cet ON t.id_teacher = cet.id_teacher
JOIN
    course_edition ce ON cet.id_course_edition = ce.id_course_edition
GROUP BY t.teacher_name
HAVING MIN(ce.workload_hours) > 60 AND MAX(ce.workload_hours) < 80;

```

Query breakdown:

1. JOIN teacher t ON course_edition_teacher cet - Links teachers to their respective course editions.

2. JOIN course_edition ce - Connects each course edition to retrieve the workload hours.
3. GROUP BY t.teacher_name - Groups results by teacher, ensuring each teacher is evaluated independently.
4. HAVING MIN(ce.workload_hours) > 60 AND MAX(ce.workload_hours) < 80 - Ensures that all courses taught by the teacher fall strictly within the range of 61 to 79 hours.

If a teacher teaches any course outside the range, they are excluded from the results.

The query uses HAVING with MIN and MAX to check the full workload range per teacher.

	TEACHER_NAME
1	Mafalda Ferreira

The output gave only Mafalda Pereira, as she is the only teacher whose courses ALL fall within the 61-79 hour range.

Query 7

The query retrieves a list of teachers, along with their categories and departments, considering only those who have never taught more than two course editions in any single academic year. It considers only teaching roles and not whether the teacher was also a director, because simply being a director does not necessarily mean that person teaches the course. Therefore, our approach only accounts for teaching roles and does not consider directorships in the count.

WITH teacher_course_counts AS (

SELECT

```

    t.teacher_name,

    tc.name AS category_name,

    d.name AS department_name,

    ce.edition_year,

    COUNT(ce.id_course_edition) AS course_count

FROM

    teacher t

JOIN

    department d ON t.id_department = d.id_department

JOIN

    teacher_category tc ON t.id_teacher_category = tc.id_teacher_category

LEFT JOIN

    course_edition_teacher cet ON t.id_teacher = cet.id_teacher

LEFT JOIN

    course_edition ce ON cet.id_course_edition = ce.id_course_edition

GROUP BY

    t.id_teacher, t.teacher_name, tc.name, d.name, ce.edition_year

)

SELECT

    teacher_name,

    category_name,

    department_name

```

FROM

teacher_course_counts

GROUP BY

teacher_name, category_name, department_name

HAVING

MAX(course_count) <= 2;

SUMMARY:

The query first counts how many course editions each teacher taught per academic year, then selects teachers who have never taught more than two course editions in any year, displaying only those who meet the criteria, along with their name, category, and department.

STEP 1: COMMON TABLE EXPRESSION TEACHER_COURSE_COUNTS:

1. JOIN tables to combine information from multiple tables:
 - JOIN department d : associating each teacher with their department
 - JOIN teacher_category tc : retrieves the teacher's category
 - LEFT JOIN course_edition_teacher cet : links teachers to the courses they've been involved in. Since it's a LEFT JOIN, teachers with no associated courses will still appear in the results, as the LEFT JOIN allows null values
 - LEFT JOIN course_edition ce : retrieves details about course editions, if a teacher has not been associated with any course editions, they will still be displayed in the result, as the count will return 0.
2. GROUP results to count courses per teacher per year:

Groups the data by teacher and academic year to calculate the number of course editions a teacher has taught per year, ensuring that each teacher appears only once in the result, with aggregated information about their course editions.

3. Counting course editions:

COUNT(ce.edition_year) AS course_count : counts how many course editions a teacher participated in per year

STEP 2: MAIN QUERY, FILTERING AND OUTPUT

1. SELECT relevant teacher details for the final output:

The query selects key columns from the CTE teacher_course_counts:

- Teacher's name : t.teacher_name.
- Teacher's category : tc.name AS category_name.
- Teacher's department : d.name AS department_name.

2. GROUPing by teacher:

Groups the results by teacher name, category and department to ensure each appears only once in the final output.

3. Filtering with the condition HAVING:

HAVING MAX(course_count) <= 2 : this clause filters the grouped results and ensures only teachers who taught no more than two distinct courses per year.

	TEACHER_NAME	CATEGORY_NAME	DEPARTMENT_NAME
1	Pedro Ferreira	Visiting Adjunct Professor	Department of Civil Engineering
2	André Fidalgo	Adjunct Professor	Department of Electrical Engineering
3	Luis Nogueira	Adjunct Professor	Department of Computer Engineering
4	Alexandre Gouveia	Adjunct Professor	Department of Computer Engineering
5	Rosa Reis	Adjunct Professor	Department of Computer Engineering
6	Nuno Silva	Coordinating Professor	Department of Computer Engineering
7	Ángelo Martins	Coordinating Professor	Department of Computer Engineering
8	Simone Morais	Visiting Adjunct Professor	Department of Chemical Engineering
9	Joaquim Alves	Associate Professor	Department of Physics
10	Mónica Vieira	Coordinating Professor	Department of Physics
11	Tiago Andrade	Associate Professor	Department of Organization and Management
12	Custódio Dias	Full Professor	Department of Electrical Engineering
13	Pedro Guedes	Assistant Professor	Department of Mathematics
14	Elisabete Esteves	Assistant Professor	Department of Geotechnical Engineering
15	José Guimarães	Assistant Professor	Department of Organization and Management
16	Mariana Oliveira	Adjunct Professor	Department of Mechanical Engineering
17	Constantino Martins	Coordinating Professor	Department of Computer Engineering
18	Sónia Pizarro	Assistant Professor	Department of Geotechnical Engineering
19	Elza Fonseca	Trainee Assistant	Department of Mechanical Engineering
20	Francisco Pereira	Coordinating Professor	Department of Electrical Engineering
21	Olga Castro	Teaching Assistant	Department of Mechanical Engineering
22	Roberto Silva	Adjunct Professor	Department of Mechanical Engineering
23	Joaquim Cunha	Associate Professor	Department of Geotechnical Engineering
24	Manuela Barroso	Associate Professor	Department of Civil Engineering
25	Paulo Proença	Adjunct Professor	Department of Computer Engineering
26	Isabel Pereira	Trainee Assistant	Department of Chemical Engineering
27	Isabel Figueiredo	Associate Professor	Department of Mathematics
28	Carla Lopes	Trainee Assistant	Department of Organization and Management
29	Mafalda Ferreira	Full Professor	Department of Organization and Management
30	Valentina Domingues	Visiting Adjunct Professor	Department of Chemical Engineering
31	Alzira Mota	Adjunct Professor	Department of Mathematics
32	Jeremias da Pereira	Associate Professor	Department of Organization and Management

The query displays ALL teachers EXCEPT three: Carlos Félix, Bruno Oliveira and Raquel Gonçalves, as they each taught three courses in one year, exceeding the specified limit. Demonstrated by the following data:

Carlos Félix in 2020/2021:

- `course_edition_teacher (id_teacher, id_course_edition) VALUES ('2', '2');`
- `course_edition_teacher (id_teacher, id_course_edition) VALUES ('2', '3');`
- `course_edition_teacher (id_teacher, id_course_edition) VALUES ('2', '4');`

Bruno Oliveira in 2019/2020

- `course_edition_teacher (id_teacher, id_course_edition) VALUES ('16', '16');`
- `course_edition_teacher (id_teacher, id_course_edition) VALUES ('16', '19');`
- `course_edition_teacher (id_teacher, id_course_edition) VALUES ('16', '20');`

Raquel Gonçalves in 2019/2020

- `course_edition_teacher (id_teacher, id_course_edition) VALUES ('24', '25');`
- `course_edition_teacher (id_teacher, id_course_edition) VALUES ('24', '26');`
- `course_edition_teacher (id_teacher, id_course_edition) VALUES ('24', '28');`

Query 8

This SQL query calculates the average final grades (`average_grade`) of students for each course, grouped by teacher and course. It only includes courses from study plans for the academic years 2021/2022 and 2020/2021.

I considered that the year 2021 mentioned in the query refers to the academic years **2020/2021** and **2021/2022**.

1. Tables Used:

- a. `student_course (sc)`: Holds student grades.
- b. `course (c)`: Contains course names.
- c. `teacher (t)`: Provides teacher names.

- d. study_plan (sp): Defines the academic year.
- 2. Filter:
 - a. Includes only data from the years 2021/2022 and 2020/2021.
- 3. Aggregation:
 - a. Groups data by teacher and course.
 - b. Calculates the average grade (AVG(sc.final_grade)).
- 4. Ordering:
 - a. Results are sorted by teacher and course names.

```
SELECT
    t.teacher_name AS teacher,
    c.course_name AS course,
    AVG(sc.final_grade) AS average_grade
FROM
    student_course sc
JOIN course c ON sc.id_course = c.id_course
JOIN course_edition ce ON ce.id_course = c.id_course
JOIN teacher t ON ce.id_teacher = t.id_teacher
JOIN study_plan_course spc ON spc.id_course = c.id_course
JOIN study_plan sp ON sp.id_study_plan = spc.id_study_plan
WHERE
    sp.schoolYear IN ('2021/2022', '2020/2021')
GROUP BY
    t.teacher_name, c.course_name
ORDER BY
    t.teacher_name, c.course_name;
```

	TEACHER	COURSE	AVERAGE_GRADE
1	Carlos Félix	Introduction to Progr	15
2	Isabel Pereira	Principles of Manage	17
3	Nuno Silva	Digital Logic Design	20
4	Pedro Guedes	Human-Computer In	5
5	Simone Morais	Project Management	12
6	Tiago Andrade	Data Science	19.75

Query 9

In this query, we would need to find the students who had a grade in the "BDDAD" course that is lower than the average grade for that course in the year 2023.

The query works by:

1. Selects student ID's for those who took the "BDDAD" course and checks if their grade is below the course's average grade in 2023;
2. Calculates the average grade for the "BDDAD" course in 2023;
3. Ensures the grades considered are only from the year 2023.

```

SELECT s.id_student
FROM student_course sc
JOIN course c ON sc.id_course = c.id_course
JOIN student s ON sc.id_student = s.id_student
WHERE c.acronym = 'BDDAD'
AND sc.final_grade < (
  SELECT AVG(sc2.final_grade)
  FROM student_course sc2
  JOIN course c2 ON sc2.id_course = c2.id_course
  WHERE c2.acronym = 'BDDAD'
  AND EXTRACT(YEAR FROM sc2.enrollment_datetime) = 2023
)
AND EXTRACT(YEAR FROM sc.enrollment_datetime) = 2023;

```


	ID_STUDENT
1	7


Query 10

The objective of this query is to list all courses for each programme and indicate whether they have prerequisites. For courses without prerequisites, the query includes an additional column titled “Observations” that specifies “No prerequisites.”

The query works by:

1. Join the programme and course tables to associate courses with programmes.
2. Use a CASE statement to check if a course has prerequisites and populate the observations column accordingly.
3. Display all courses, indicating whether they have prerequisites, and sort the results by programme and course name.

```
SELECT p.programme_name, c.course_name, c.prerequisites,
CASE
  WHEN c.prerequisites = 'no' THEN 'No prerequisites'
  ELSE '' -- Deixa a célula vazia para cursos com pré-requisitos
END AS observations
FROM
  programme p
JOIN
  course c ON p.id_programme = c.id_course
ORDER BY
  p.programme_name, c.course_name;
```

	PROGRAMME_NAME	COURSE_NAME	PREREQUISITES	OBSERVATIONS
1	Advanced Machining	Quantum Mechanics	yes	
2	Advanced Structural	Computer Networks	yes	
3	Agile Software Devel	Computer Graphics	yes	
4	Artificial Intelligence I	Software Engineering	yes	
5	Automotive Engineer	Algorithms	yes	
6	BigData Decision Ma	Project Management	yes	
7	Biomedical Engineer	Database	yes	
8	Bioresources	Introduction to Progr	no	No prerequisites
9	Chemical Engineerin	Web Development	yes	
10	Civil Engineering	Physics I	no	No prerequisites
11	Computer Engineerin	Chemistry I	no	No prerequisites
12	Database Administra	Machine Learning	yes	
13	Digital Construction	Principles of Manage	no	No prerequisites
14	Digital Construction I	Human-Computer In	yes	
15	Electrical Engineerin	Civil Engineering	no	No prerequisites
16	Energy Efficiency in E	Microeconomics	no	No prerequisites
17	Energy, Efficiency an	Thermodynamics	yes	
18	Georesources, Const	Cybersecurity	yes	
19	Geotechnical Engine	Laboratory-Project	yes	
20	Industrial Automatiz	Artificial Intelligence	yes	
21	Industrial Engineerin	Database Systems	yes	
22	Information Security	Macroeconomics	yes	
23	Logistics Supply Cha	Biology	no	No prerequisites
24	Metrology, Instrum	Data Science	yes	
25	Process and Operatic	Economics for Engin	no	No prerequisites
26	Supply Chain Manag	Electrical Circuits	no	No prerequisites
27	Sustainable Energies	Digital Logic Design	no	No prerequisites
28	Switch	Mathematics I	no	No prerequisites
29	Systems Engineering	Linear Algebra	no	No prerequisites
30	Telecommunications	Discrete Mathematic	yes	

Query 11

The objective of this query is to identify students who enrolled in courses before 2022, after 12:30 PM, and had more than two enrollments on the same enrollment date.

The query works by:

1. Joining the student and student_course tables to retrieve each student's enrollment details.

2. Filtering for enrollments that occurred:
 - a. Before the year 2022.
 - b. After 12:30 PM based on the enrollment timestamp.
3. Grouping by the student's name and the exact enrollment datetime to calculate enrollment counts for each timestamp.
4. Using the HAVING clause to ensure only groups with more than two enrollments are included.
5. Formatting the output to display the enrollment date, time, day of the week, and month in a user-friendly format.
6. Sorting the results by enrollment date (descending) and enrollment time (ascending).

SELECT

```
s.student_name AS Name,  
  
TO_CHAR(sc.enrollment_datetime, 'YYYY-MM-DD') AS Enrollment_Date,  
  
TO_CHAR(sc.enrollment_datetime, 'HH24:MI') AS Enrollment_Time,  
  
TO_CHAR(sc.enrollment_datetime, 'Day') AS Day_Of_Week,  
  
TO_CHAR(sc.enrollment_datetime, 'Month') AS Month
```

FROM

```
student_course sc
```

JOIN

```
student s ON sc.id_student = s.id_student
```

WHERE

```
EXTRACT(YEAR FROM sc.enrollment_datetime) < 2022  
  
AND TO_CHAR(sc.enrollment_datetime, 'HH24:MI') > '12:30'
```

GROUP BY

s.student_name, sc.enrollment_datetime

HAVING

COUNT(sc.id_course) > 2 -- More than 2 enrollments

ORDER BY

TO_CHAR(sc.enrollment_datetime, 'YYYY-MM-DD') DESC,

TO_CHAR(sc.enrollment_datetime, 'HH24:MI') ASC;

	NAME	ENROLLMENT_DATE	ENROLLMENT_TIME	DAY_OF_WEEK	MONTH
1	Nuno Cunha	2021-03-01	15:30	Monday	March
2	Lucas Pereira	2020-03-01	15:30	Sunday	March
3	Pedro Costa	2020-03-01	15:30	Sunday	March
4	Ana Oliveira	2020-03-01	15:30	Sunday	March
5	João Silva	2020-03-01	15:30	Sunday	March
6	Maria Sousa	2020-03-01	15:30	Sunday	March

Query 12

For the first semester of the 2022-2023 academic year, list the classrooms where lecture classes were taught most often.

1. Focus on the first semester of 2022-2023 school_year.
2. Select only lecture classes (course_class_type "T").
3. Group by classroom and count occurrences.
4. Sort results by the order DESC;

SELECT

cl.id_classroom AS classroom_id,

COUNT(cc.id_course_class) AS total_classes

FROM

course_class cc

JOIN

```

        classroom cl ON cc.id_classroom = cl.id_classroom
JOIN
        course_edition ce ON cc.id_course_edition = ce.id_course_edition
JOIN
        course co ON ce.id_course = co.id_course
JOIN
        study_plan_course spc ON co.id_course = spc.id_course
JOIN
        study_plan sp ON spc.id_study_plan = sp.id_study_plan
JOIN
        course_class_type cct ON cc.id_course_class_type =
cct.id_course_class_type
WHERE
        sp.schoolYear = '2022/2023'
        AND spc.semester = 1
        AND cct.description = 'T'
GROUP BY
        cl.id_classroom
ORDER BY
        total_classes DESC;

```

	CLASSROOM_ID	TOTAL_CLASSES
1	E490	6
2	B109	4
3	B106	2
4	D106	1

Query 13

In the Query nº13, for each professor in the computer engineering department, we need to list the classrooms where they taught classes, as well as the type of class and the respective classes. Only professors who taught courses in the second

semester of the 3rd curricular year are the ones that we are the only ones we will take into account.

In this query, we will look to list only the professors that taught course in the second semester of the 3rd curricular year and that only belong to the computer engineering department. And GROUP it with the order for Teacher, Classroom, Class_Description and Course.

```
SELECT
t.teacher_name AS teacher,
cc.id_classroom AS classroom,
    cct.description AS type,
    c.id_course AS class
FROM
teacher t
JOIN
department d ON t.id_department = d.id_department
JOIN
course_class cc ON t.id_teacher = cc.id_teacher
JOIN
course_edition ce ON t.id_teacher = cc.id_teacher
JOIN
    course c ON ce.id_course = c.id_course
JOIN
    course_class_type cct ON cc.id_course_class_type = cct.id_course_class_type

JOIN
study_plan_course spc ON c.id_course = spc.id_course
JOIN
course_edition ce ON cc.id_course_edition = ce.id_course_edition
JOIN
couse c ON ce.id_course = c.id_course
```

```

JOIN
Course_class_type cct ON cc.id_course_class_type = cct.id_course_class_type
JOIN
Study_plan_course spc ON c.id_course = spc.id_course
WHERE
D.name = 'Department of Computer Engineering'
AND c.year_in_programme = 3
    AND spc.semester = 2;

```

	TEACHER	CLASSROOM	TYPE	CLASS
1	Nuno Silva	H340	TP	14

Query 14

In Query nº14 we need to list the department names and the teachers who have classes in the classrooms “B109” and “B106” on Mondays, Wednesdays and Fridays from 8 a.m. to 1 p.m. and haven't been absent in 2 months.

Firstly, we join the tables “teacher” and “course_class” using “id_teacher” to link each teacher with their course class. Then, we join the “course_class” to “weekly_timetable” to get the schedule for each class. Lastly, we join “teacher” with “department” to associate teachers with their respective departments.

We select the teacher_name (t.teacher_name) and department_name (d.name).

Now we need to filter the classrooms through the “id_classroom” that coincides with “B109” or “B106”. Then we filtrate the weekdays using to_char(cc.date_start_time, 'DY', 'NLS_DATE_LANGUAGE=ENGLISH') to extract the day of the week from the cc.date_start_time and compare to (MON, WED, FRI). We also need to extract the time to ensure that the course_class starts between 8PM and 1PM using EXTRACT(HOUR FROM cc.date_start_time).

To check if a teacher wasn't “Absent” within the last 2 months we check the “teacher_attendance” table for the “Absent” status based on the condition on the course_class.date_start_time.

Lastly, we sort the result by department name and teacher name.

```
SELECT
    t.teacher_name,
    d.name AS department_name
FROM
    course_class cc
JOIN
    teacher t ON cc.id_teacher = t.id_teacher
JOIN
    department d ON t.id_department = d.id_department
JOIN
    weekly_timetable wt ON wt.id_weekly_timetable = cc.id_weekly_timetable
WHERE
    cc.id_classroom IN ('B109', 'B106')
    AND TO_CHAR(cc.date_start_time, 'DY', 'NLS_DATE_LANGUAGE=ENGLISH') IN
('MON', 'WED', 'FRI')
    AND EXTRACT(HOUR FROM cc.date_start_time) >= 8
    AND EXTRACT(HOUR FROM cc.date_start_time) < 13
    AND NOT EXISTS (
        SELECT 1
        FROM teacher_attendance ta2
        WHERE ta2.id_teacher = t.id_teacher
        AND ta2.status = 'Absent'
        AND cc.date_start_time >= ADD_MONTHS(SYSDATE, -2))
    AND NOT EXISTS (
        SELECT 1
        FROM teacher_attendance ta
        WHERE ta.id_teacher = t.id_teacher
        AND ta.status = 'Absent'
        AND ta.id_course_class = cc.id_course_class
```


)
ORDER BY
d.name, t.teacher_name;

	TEACHER_NAME	DEPARTMENT_NAME
1	Joaquim Cunha	Department of Geotechnical Engineering
2	Jeremias da Pereira	Department of Organization and Management
3	jose Guimarães	Department of Organization and Management

Query 15

This query aims to identify students with the highest number of absences in the second year of their programme (2nd year courses) during the months of June and July (we considered those as being the last two months always), and check if they are enrolled in courses with a higher failure rate than the course LABPROJ1 in the academic year 2019/2020.

WITH

labproj1_failure_rate AS (

SELECT

course.id_course,

course.course_name,

COUNT(CASE WHEN student_course.final_grade < 10 THEN 1 END) /
NULLIF(COUNT(*), 0) AS failure_rate

FROM

student_course

JOIN course ON student_course.id_course = course.id_course

JOIN course_edition ON course.id_course = course_edition.id_course

```

WHERE

    course.course_name = 'LABPROJ1'

    AND course_edition.edition_year = '2019/2020'

GROUP BY

    course.id_course, course.course_name

),

high_failure_courses AS (

    SELECT

        c.id_course,

        c.course_name,

        COUNT(CASE WHEN sc.final_grade < 10 THEN 1 END) / NULLIF(COUNT(*), 0)

    AS failure_rate

    FROM

        student_course sc

    JOIN course c ON sc.id_course = c.id_course

    JOIN course_edition ce ON c.id_course = ce.id_course

    WHERE

        ce.edition_year = '2019/2020'

    GROUP BY

        c.id_course, c.course_name

    HAVING

        COUNT(CASE WHEN sc.final_grade < 10 THEN 1 END) / NULLIF(COUNT(*), 0)

    > (

```

```

        SELECT failure_rate

        FROM labproj1_failure_rate

    )

),

student_absences AS (

    SELECT

        sa.id_student,

        st.student_name,

        c.course_name,

        COUNT(*) AS total_absences

    FROM

        student_attendance sa

    JOIN student st ON sa.id_student = st.id_student

    JOIN course_class cc ON sa.id_course_class = cc.id_course_class

    JOIN course_edition ce ON cc.id_course_edition = ce.id_course_edition

    JOIN course c ON ce.id_course = c.id_course

    WHERE

        c.year_in_programme = 2

        AND EXTRACT(MONTH FROM cc.date_end_time) IN (6, 7)

        AND sa.status = 'Absent'

    GROUP BY

        sa.id_student, st.student_name, c.course_name

```

```

),
max_absences AS (
    SELECT
        id_student,
        student_name,
        MAX(total_absences) AS highest_absences
    FROM
        student_absences
    GROUP BY
        id_student, student_name
)
SELECT
    sa.student_name,
    sa.total_absences
FROM
    student_absences sa
JOIN max_absences ma ON sa.id_student = ma.id_student
WHERE
    sa.total_absences = ma.highest_absences
    AND sa.course_name IN (SELECT course_name FROM high_failure_courses);

```

Steps:

1. Calculate the failure rate for the course LABPROJ1 in the academic year 2019/2020:

The subquery `labproj1_failure_rate` calculates the failure rate for the course LABPROJ. Counts the number of students who failed (`final_grade < 10`) and divides it by the total number of students in that course, using `NULLIF` to avoid division by zero. It filters to include only the academic year 2019/2020, and groups the results by `id_course` and `name`.

2. Retrieve courses with a higher failure rate than the course LABPROJ1:

The subquery `high_failure_courses` calculates the failure rate for all courses in the same academic year (2019/2020). It counts the number of failing students for each course and calculates the failure rate. The `HAVING` clause compares each course's failure rate with the LABPROJ1 one, selecting only those with a higher.

3. Retrieve absences of second-year students in the months of June and July:

The subquery `student_absences` counts student absences for course in the 2nd year of the programme, it filters the records to include only those where students were marked as Absent in the last two months (June and July), irrespective of the year.

4. Find the student with the highest number of absences:

The subquery `max_absences` identifies the student(s) with the maximum number of absences (`MAX(total_absences)`). It groups the data by `id_student` and `student_name` to get the students with the highest absence count.

5. Display results with the highest absences and those enrolled in courses with a higher failure rate:

The final `SELECT` statement retrieves the student name and total absences. It joins the list of students with the highest absences (`max_absences`) and filters those whose courses have a higher failure rate than LABPROJ1, as determined before in STEP 2.

	STUDENT_NAME	TOTAL_ABSENCES
1	João Silva	1
2	Maria Sousa	1
3	Pedro Costa	1
4	Ana Oliveira	1

Query 16

This SQL query creates a view called StudentsAboveAverageAttendance. It identifies students whose total number of attendances (marked as "Present") in a class exceeds the average attendance for that class.

1. Subquery:
 - a. Calculates the average attendance (avg_count) per class by dividing the total number of "Present" statuses by the number of distinct students in the class.
2. Main Query:
 - a. Counts the total "Present" statuses (student_present_count) for each student.
 - b. Joins this count with the class average attendance (avg_count).
3. Filter (HAVING):
 - a. Only includes students whose attendance is greater than the class average.
4. Result:
 - a. Returns the student name (student_name), their attendance count (student_present_count), and the class average attendance (class_avg_count).

```
CREATE VIEW StudentsAboveAverageAttendance AS
SELECT
s.student_name,
COUNT(CASE WHEN sa.status = 'Present' THEN 1 END) AS student_present_count,
```

```

avg_attendance.avg_count AS class_avg_count
FROM
student s
JOIN
student_attendance sa ON s.id_student = sa.id_student
JOIN
(
SELECT
sa.id_course_class,
SUM(CASE WHEN sa.status = 'Present' THEN 1 ELSE 0 END) * 1.0 /
COUNT(DISTINCT sa.id_student) AS avg_count
FROM
student_attendance sa
GROUP BY
sa.id_course_class
) avg_attendance ON sa.id_course_class = avg_attendance.id_course_class
GROUP BY
s.student_name, avg_attendance.avg_count
HAVING
COUNT(CASE WHEN sa.status = 'Present' THEN 1 END) >
avg_attendance.avg_count;

```

--Use this select to test.

```

SELECT * FROM StudentsAboveAverageAttendance;

```

	STUDENT_NAME	STUDENT_PRESENT_COUNT	CLASS_AVG_COUNT
1	Luis Sousa	1	0.75
2	Miguel Ribeiro	1	0.75
3	Ana Martins	1	0.75
4	Luis Sousa	1	0.25
5	André Almeida	1	0.5
6	Sara Gomes	1	0.6666666666666666
7	Maria Ferreira	1	0.6666666666666666
8	Tiago Pereira	1	0.5
9	Rafael Santos	1	0.5
10	Pedro Costa	1	0.3333333333333333
11	Lucas Pereira	1	0.2

Conclusion

In the scope of the project developed, it was possible to practically apply the fundamental concepts of relational database modeling and implementation. This work was divided into three main parts: the creation of a relational model, its implementation in a database, and the development of SQL commands to perform queries on the implemented database.

During the development of the relational model, the relevant entities, attributes, and relationships were identified, ensuring that the model complied with the Third Normal Form (3NF). This normalization guaranteed the elimination of redundancies as well as the integrity and consistency of the data.

Primary and foreign keys were defined, along with the cardinalities of the relationships between the entities. Furthermore, integrity constraints that could not be directly represented in the model, such as specific business validations, were thoroughly documented and justified.

The practical implementation included the creation of tables and the application of customized constraints. Appropriate data types were used, and constraints such as:

- Field validation (for example, minimum and maximum value limits);
- Validation of Allowed Values (Check Constraint);
- Uniqueness and foreign key constraints to ensure data consistency between tables.

The SQL queries presented were constructed to demonstrate the effectiveness of the model and the ease of extracting useful information to support decision-making. These queries ranged from basic data extraction operations to more complex ones involving multiple tables and specific conditions.

Throughout the project, all decisions were substantiated to ensure the model was robust, aligned with the requirements of the domain in question and scalable. Scalability, ensures that the database structure could efficiently handle increasing amounts of data and adapt to future requirements without significant redesign, making it capable of supporting evolving business needs.

Overall, this project demonstrated the importance of a well-structured model and its careful transition to practical implementation in a database. The project reinforced the importance of normalization and documentation to ensure reliable and high-quality data systems.