Project Report for Course:

DV1663 - Databasteknik

2023-08-13

Database Project

Johan Nilsson, Nikola Antic

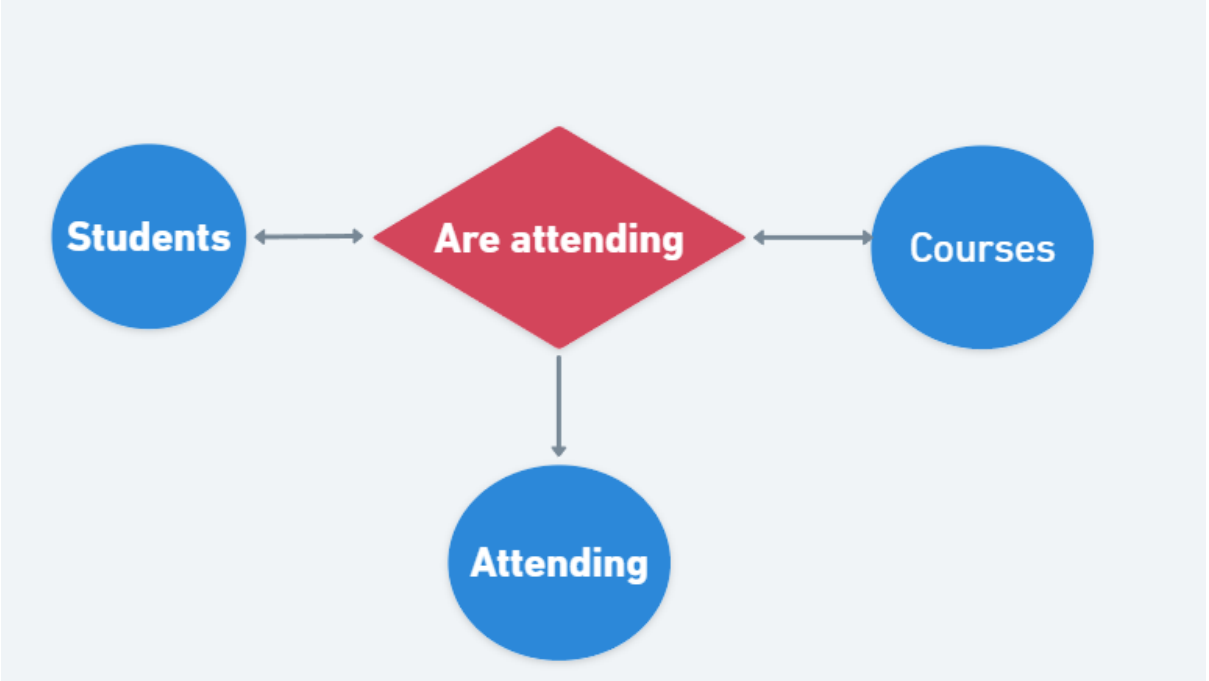
Faculty of Computing
Blekinge Institute of Technology
Sweden, Karlskrona

Problem & Idea

A problem that we have had ourselves and that the majority of the students have is that it is really hard to calculate what you need for “Förkunskapskraven” in order to register for certain courses. These are often very hard to see exactly what you need, and when it comes to specific requirements such as “You need X hp in the area of Computer Science”, it is very hard to know what courses count as which, especially since it can differ a lot. In order for a student to get a complete overview of their education situation and what courses they are ineligible for they have to book and attend meetings, which is not ideal for a student since they usually just want to focus on studying, and tend to be a little bit lazy when it comes to meetings. This system would help students and allow them to structure their education correctly in order to be able to register for all the courses in their program in an easy and efficient way which would hopefully make more students aware of their education situation, since we know that many students don’t want to schedule a time with Studievägledningen or send email in order to get this information.

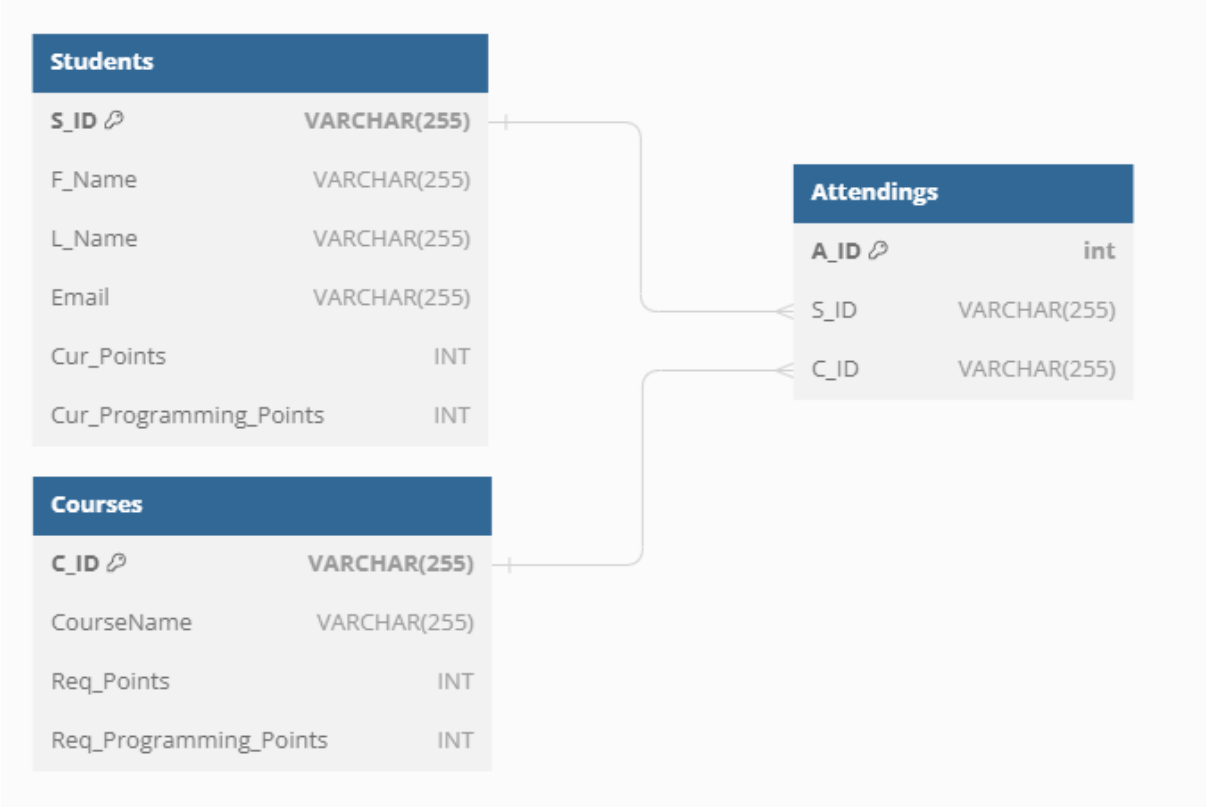
The system will be created with a web interface, where you can log in with your student credentials which will integrate your information from the database. The system will allow the user to apply for a course, drop out of a course, see all of the courses that they are currently reading and also let you search for specific courses.

Logical model



We don’t have a huge project so the er diagram is not that big either, attributes and ex. one to many relations will be discussed in the next section (so don't mind the arrows) . To explain this diagram Students are attending courses and courses are attended by students. We were not sure on how to describe this in the picture, but the relationship is basically a table called attendings which keeps track on who is attending which course and which course is attended by which student. So there is no direct relation between the tables Students and Courses in practice as we will see in the schema.

SQL SCHEMA



This is the translated schema from the ER diagram including the attributes and the relation arrows. We are using three tables where both Students and Courses relate to the attendings table because information from those two are stored there. There is no direct relation between the student and the course table, but they are related through the attendings table. Students can attend many courses but each specific attending can only have one studentID (because one student can’t attend the same course twice at the same time. Many courses can be attended, but once again a student can’t attend the same course at the same time.

This is ensured through the primary keys (S_ID) and (C_ID) from Students and Courses and as they are referenced as foreign Keys in the Attendance table ensuring referential integrity. S_ID, C_ID and A_ID were all chosen as primary keys for each table as they are unique and therefore suitable as a primary key. Every other attribute is just necessary information and enough information to use when working on a project like this.

SQL queries

For this project, we had to use at least five queries. At least two of the queries had to deal with data from more than one table. We also had to make use of SQL JOIN, Aggregation/Grouping and two of the following: Triggers, Procedures and Functions.

There we will go through all of the queries used for our project, together with an explanation and motivation of each query.

Query 1:

```
"""select *from Students where S_ID = %s"""
```

This query was used in the login phase of the system, where the student had to enter their student ID in order to continue. This query selects all information from students where the S_ID is the same as they entered into the textbox. The S_ID got saved into a global variable in order to keep track of which students information that would be displayed later.

Query 2:

```
"""SELECT * FROM courses WHERE C_ID = %s"""
```

This query was used in the homepage in the system, when the student searched for a course in a big search box. This query selects all the information from courses where the CourseID (C_ID) is the same as the one they entered. This was also used to save coursename into a global variable which allowed us to display the name on the course page.

Query 3:

```
("SELECT S_ID, F_Name, L_Name FROM Students WHERE S_ID = %s", (s_id,))
```

This query selects the student ID, First name, Last name from the particular student which is using the system. This information was used together with

query 4, in query 5 to insert all the necessary information into the table Attendings when a student applied for a course.

Query 4:

```
"SELECT CourseName FROM Courses WHERE C_ID = %s", (c_id,))
```

This query selects the CourseName from Courses for the particular course in question, which the student applied to. This information was later used in the next query together with query 3.

Query 5:

```
"""
                    INSERT INTO Attendings (A_ID, S_ID, F_Name, L_Name,
C_ID, CourseName)
                    VALUES (%s, %s, %s, %s, %s, %s)
                """
```

This query inserts A_ID, S_ID, F_Name, L_Name, C_ID and CourseName into the table Attendings. This information makes it possible for us to add students into attendings, which is a crucial part of the system.

Query 6:

```
("DELETE FROM Attendings WHERE C_ID =%s AND S_ID =%s", (c_id,s_id))
```

This query is used when a student chose to drop out of a class, which means that their attending for that course needs to be deleted from the attendings. This deletes all information about the student, for that particular course, and therefore removes them from the Attending table for that course, while they still remain there for their other courses.

Query 7:

```
("UPDATE Attendings SET A_ID = A_ID - 1")
```

This query updates the Attending ID in the Attendings table. This query happens right after query 6, after a student has been removed from the table. The query updates all A_ID to A_ID minus 1, which makes sure that there is no duplicated occurring while inserting a new attending later.

Query 8:

```
"""SELECT courses.CourseName FROM courses
                            INNER JOIN attendings ON courses.C_ID =
attendings.C_ID
                            WHERE attendings.S_ID = %s"""
```

This query is used when looking what courses that a student is currently attending. This is done by selecting the coursename, and then inner join attendings on courses.C_ID = Attendings.C_ID, where attendings.S_ID is the S_ID in question.

Query 9: """

```
        SELECT S_ID, F_Name, L_Name, Cur_Points
        FROM Students
        WHERE Cur_Points = (
            SELECT MAX(Cur_Points) FROM Students
        )
    """
```

This query is used to display the student which has the most points at the school. This is done by selecting all the information which we wanted to display, and using MAX() to get the student which had the most points.

Procedure 1:

Procedure that checks if the students have enough points to attend a course. Join the Student and Course table together and count the rows where Student ID course ID is equal to the input parameters and if the students total points and programming points is greater or equal to the total required points and required programming points of the desired course. If it is then it returns a 1 (you can attend the course) else 0 (you cant attend the course)

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `Check_Points`(IN sid
varchar(255), IN cid varchar(255), OUT result INT)
BEGIN
Declare var int;

select COUNT(*) into var from Students CROSS JOIN Courses
 where S_ID = sid AND C_ID = cid AND Students.Cur_Points >=
Courses.Req_Points AND Students.Cur_Programming_Points >=
Courses.Req_Programming_Points;

 If var > 0 then
      SET result = 1;

ELSE
      SET result = 0;

END IF;
```

```
END
```

Procedure 2:

Procedure that checks if a student is attending a course. It joins all the three tables together and counts the rows where the student ID and course ID is equal to the input parameters. If a student is attending the desired course the total_count will be 1 and 0 if they are not attending a course. A 0 or 1 is returned.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `Student_Attend_Check`(IN
student varchar(255), IN cid varchar(255), OUT result INT)
BEGIN
declare total_count int;
SELECT COUNT(*) INTO total_count FROM Attendings
    LEFT JOIN Students ON Attendings.S_ID = Students.S_ID
    LEFT JOIN Courses ON Attendings.C_ID = Courses.C_ID
   WHERE Students.S_ID = student AND Courses.C_ID = cid;

IF total_count >= 1 then
SET result = 1;

ELSE
SET result = 0;

END IF;

END
```

Function 1:

Function that count the rows in the attendings table, which was used to know how the attending ID:s The attendance table was going to be rearranged. (This we forgot to mention in the video)

```
CREATE DEFINER=`root`@`localhost` FUNCTION `count_rows`() RETURNS int
BEGIN
DECLARE number int;

SELECT COUNT(*) into number from Attendings;

RETURN number;
END
```

Appendix
Supplemental Video: https://youtu.be/YdkA_JnwejI
Github: https://github.com/johannilsson8/Databas

Changelog:
2023-04-22: Both of us created the E/R Diagram
2023-05-08: Adjusting to Feedback
2023-05-10: Both of us created the SQL Schema

2023-05-15 to 2023-05-26: Both of us worked on the code together using Visual Studio Code Live Share, therefore there is no specific changelog since all of the code was created together in the same document, and the liveshare did not give an opportunity to work one part of the code each at the same time.

2023-05-26: Johan: Problem & Idea, SQL Queries explanation, Appendix.
Nikola: Procedures explanation, SQL Schema, Logical Model.

2023-08-13: Both, Removed redundancy and updated diagrams