dDB Assignment 1, DA6 Group 03

Christian Zhuang-Qing Nielsen[1], Nikolai Straarup[2], Peter Rosendal[3], og Thomas Øther Rasmussen[4]

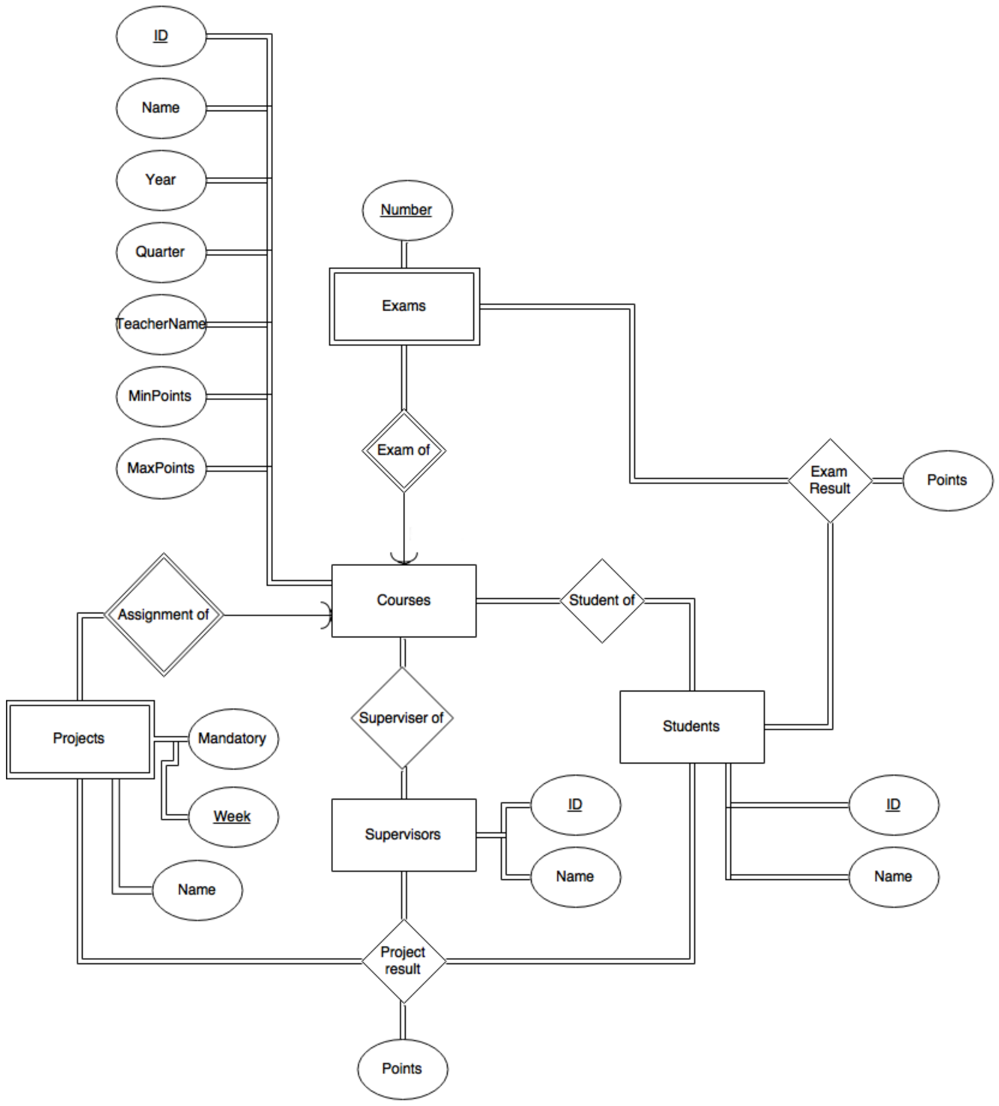
[1]201504624, christian@czn.dk
[2]201505505, nikolaistraarup@gmail.com
[3]201508600, peter_rosendal@hotmail.com
[4]201505340, thomas@tksf.dk

September 5, 2016

1 Introduction

The purpose of this assignment is to design an Entity-Relationship diagram for the administration of courses and exams in a computer science department at an undisclosed university. Doing this requires giving thought to what entity sets are required and what attributes these need as well as their interconnectivity using relationships. Furthermore, as the assignment description is rather vague, we must make some assumptions and argue for these as well as our choices regarding (weak) entity sets and types of relationships. Finally we must convert our diagram to a relational database schema using the algorithm from the textbook while explaining the individual steps.

2 Create E/R Diagram for the Computer Science Department at a University



3 Explaining Modelling Choices

3.1 Problem Statement and Assumptions

In our model we must be able to make queries based on how we imagine the database to be used. This involves a query of whether a student has submitted all mandatory exercises, how many points a student has acquired through a course, etc. To aid this, we have made numerous assumptions regarding the design of the database. These are as follows:

We assume that projects are a generalisation of hand-ins (containing all the different types of assignments). Furthermore, we assume that the university uses a points-based system, where the final grade of the student is calculated by the amount of points gained from the projects as well as the exam. Next, we assume that there is only one teacher for each course. Originally we had the teachers as an entity set but with a many-to-one restriction on the multiplicity of the *Courses-Teacher* relation. We found that the conditions were met for it to be merged with the *Courses* entity set, and thus it became an attribute of it.

Other assumptions are that each course has at most *one* project each week, and that the attribute of *Project result* also contains whether or not each mandatory project is accepted and thus whether or not a given student may go to the exam.

3.2 Our Solution

We have chosen 5 entity sets: *Projects*, *Courses*, *Exams*, *Students* and *Supervisors*, where most revolves around the *Courses*. A course has a number of projects, while a project is related to exactly one course. This explains our chosen many-one, rounded arrow relation from *Projects* to *Courses*.

Each course is given a unique ID, a name, year and a quarter. The teacher's name is an attribute of the course, since we decided that the only data concerning the teacher to be used, is their name. This should be provided when they are assigned to a course. A course carries 2 *Points*-attributes: *MinPoints* and *MaxPoints*. *MinPoints* represents the minimum number of points required to pass the course, whereas *MaxPoints* represents the maximum number of points available. This is used along with the average student score to calculate a grade for each student.

3.3 Projects and Points

The most complex problem though, is how to handle points given when projects are corrected. The points are obviously related to the project being corrected, but the project is, in turn, related to both the course, and the student who submitted the hand-in. Also, we assume that a supervisor corrects the project, meaning one more relation to address. We solve it by having a 3-way many-many-many relationship between Projects, Supervisors and Students with no restriction on multiplicity. For a student and a project, we believe it would be useful to be able to determine the supervisor who corrected the project. This relation carries the attribute *Points*, which describes the points given for the project.

3.4 Exams

We have chosen *Exams* and *Projects* to be weak entity sets, as they are dependent on the *CourseID*-attribute as part of their key, this makes them both easy to identify without adding unnecessary identification-attributes. Another main problem is how *Exams* relate to the *Courses* and *Students*. There is one exam per course, however a student can take an exam more than once, in case they failed. In our model each exam attempt is considered a new exam for the course, its key *Number* describing the attempt number. This relation doesn't go the other way round: an exam can certainly not exist without a course. This gives us a many-one relationship from *Exams* to *Courses*. As we did in the *Project result*-relation, we also store the *Points* in the *Exam Result*-relation. The multiplicity of the *Exams*-*Courses* relation is a many-one relation requiring at least one course to function properly. This is similar to the *Projects*-*Courses* many-one relation.

3.5 Using the Database Example

A sensible query is to decide whether a student has submitted all mandatory projects and so, is allowed to attend the exam. Such a query uses the *StudentID*, and the *CourseID* to list all of the course projects. Of these projects only the mandatory ones are selected. Through the relation *Project result* we view the score for each of the mandatory projects. If the score of one of these is zero, the student has failed to hand in the mandatory projects and thus cannot attend the exam.

4 Generating a Database Schema

When converting an E/R Diagram to a database schema there is a specific algorithm to follow. First, each entity set is converted to a relation (a *relation* in this instance, is a table in the database schema) containing all the attributes of the entity set. Next all the relationships are converted to relations, where each relation contains all the key attributes from the related entity sets and the attributes of the relationship itself.

Example 1: The entity set *Students* has two keys *Name* and *ID*, so the studens relation will become: Students(StudentID, StudentName). We add *Student* infront of each attribute in order to avoid confusion with other attributes with the same name.

The weak entity sets are converted to relations as well, but instead of just containing it's own attributes, it also contains the key attributes of the entity sets in any supporting relationships (it is shown as a double lined rhombus). The supporting relationships are not converted to relations.

Example 2: The entity set *Projects* is a weak one, and the resulting relation contains it's own attributes along with the keys from the entity sets in the supportive relationship *Assignment of.* The relation therefore becomes: Projects(CourseID, Week, Mandatory).

For all non-supporting relationships which include a weak entity set, the resulting relation needs to contain the keys of the weak entity set and the keys of any supporting relationships of the weak entity set.

Example 3: The relationship *Exam result* contains the weak entity set *Exams* and the entity set *Students*, it therefore uses the keys from Students and the keys from Exams and the keys from the supportive relationship's entity set *Courses*. The resulting relation therefore is: Exam-result(CourseID, Number, StudentID, Points).

4.1 Entity sets

```
Courses(CourseID, CourseName, Year, Quarter, TeacherName,
    MinPoints, MaxPoints)
Projects(CourseID, Week, Mandatory)
Supervisors(SupervisorID, SupervisorName)
Students(StudentID, StudentName, PassedCourse)
Exams(CourseID, Number)
```

4.2 Relations

```
Student-of(CourseID, StudentID)
Superviser-of(SupervisorID, CourseID)
Project-result(courseID, Week, SupervisorID, StudentID, Points)
Exam-result(CourseID, Number, StudentID, Points)
```