Basics of database systems

Project – Database design

Lappeenranta-Lahti University of Technology LUT
Software Engineering

Basics of database systems
Spring 2023

Janni Timoskainen
Leevi Laitala

TABLE OF CONTENTS

1 DEFINITION

Project Student -database is developed for a school who wants to provide a service for their students, where details of courses, their teachers and students themselves are stored and utilized. In the database, students can also store their employment status, which also enables the school to generate relevant metrics and assess their overall performance. Most importantly, the school staff are the authors that manage this database so that crucial information as id's do not leak to people that don't need to know them.

So, the database holds information/data of the students and teachers like names, emails, IDs and birthdays. There's also data of the company the students may work in and in what projects' do they contribute into. As it's a student database there's also class and university information.

The student database can be accessed via a python interface. Through this interface a user can use five already made queries, but they can also modify the database. The modifications can be only made to the student information and for example, they cannot change their id's. There's option for searching, deleting and updating the student table.

It's also important that the search can be done easily and most importantly that the student information can be searched in a logical way. That's also a part reason why we made an index that sorts the student's information by their last name.

The following queries to the database are implemented:

(1) List the information of specific student/teacher

(2) List contact information (first- and last name and email) of either students or teachers

(3) List all the students (student id, first- and last name) and their class that they attend to

(4) If a student works in company to which project(s) do they contribute into

(5) List all the teachers from a chosen university

2 MODELING

2.1 Concept model

The Figure 1 is the ER model of the designed database. There's six entities and six relationships. The entities are Student, Class, Teacher, University, Company and Project. There's 5 one-on-many (1:N) relationships and 3 many-to-many (N:M) relationships in the ER model. The N:M relationships are between 'student' and 'class', 'class' and 'university' and between 'class' and 'teacher'. This way we made sure that there can be multiple students in the same class and also multiple teachers can teach same class. Also, we wanted to make sure that multiple courses can be in many universities etc. All the other relationships are 1:N as said before and they also have, though a bit self-explanatory reasons.

The 'course_ID 'was taken off when modifying this ER Model to Relational model. On the other hand, we managed to do quite well the transformation from the ER model to the Relational model without any other changes. Of course, we considered the relationships time after time and changed them so that they work together well and so that the database can be implemented logically. So basically, the RM is quite the same as this ER model.

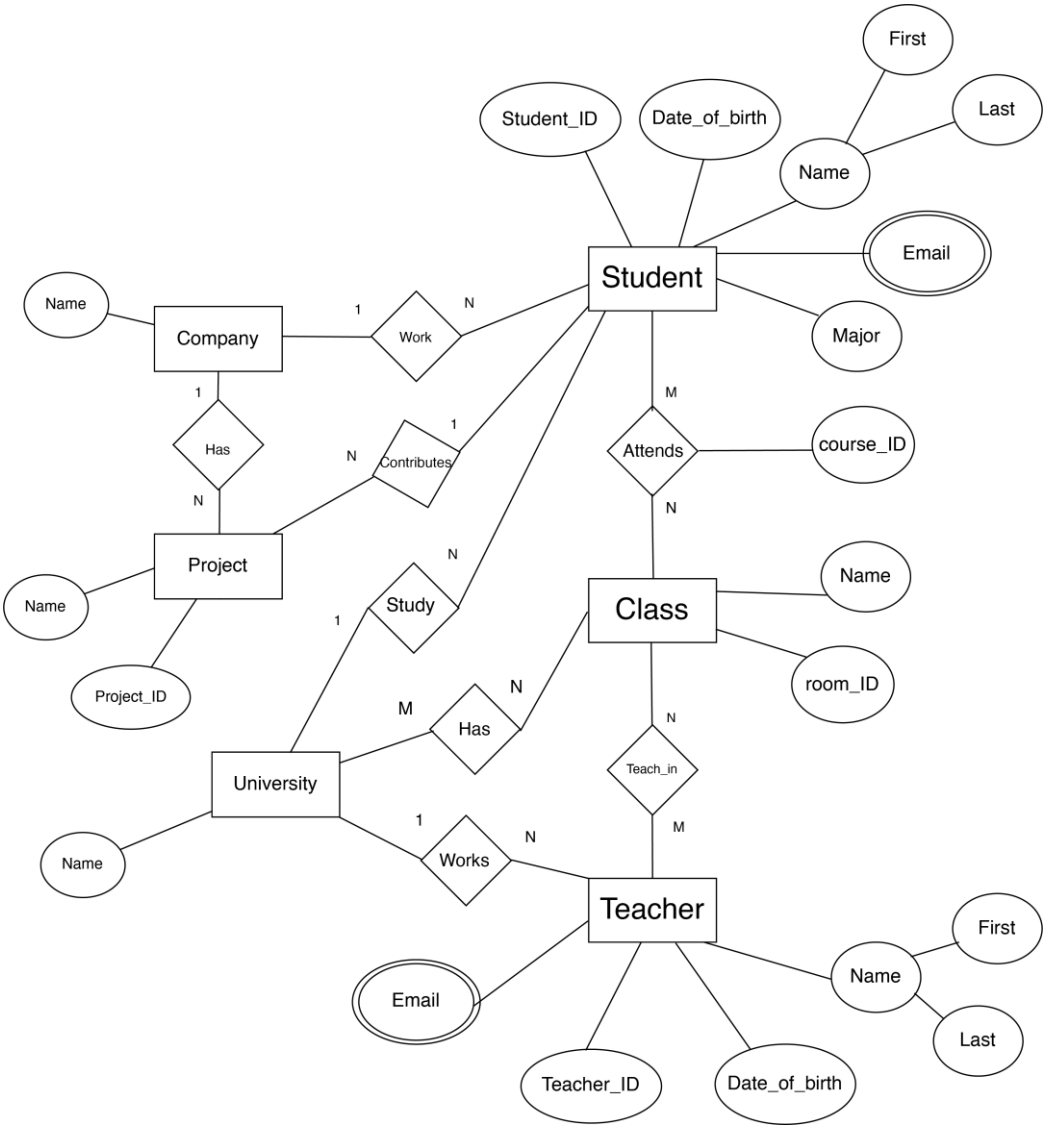

Figure 1: ER model

2.2 Relational model

Figure 2 is the RM (Relational Model) model that's created based on the ER model. As said before there wasn't much of changes made when creating this RM model of the ER model. We left the course id out and added a 'university name' foreign key to the teacher table. University has foreign keys from student and teacher table, and we also added a foreign key 'class name' to the university table. Over all these modifications that we're made are also added to the ER model, so these models don't really differ from each other.

The entities stayed the same as in the ER model. The foreign keys that are shown in this RM model helps to understand the database a bit more. This RM model represents better the relationships but also the fact that the student needs to work in a company in order to have a project that they're contributing into. We also decided that almost all of the information is crucial to, for example, a student so the foreign keys or other keys mostly cannot be null.

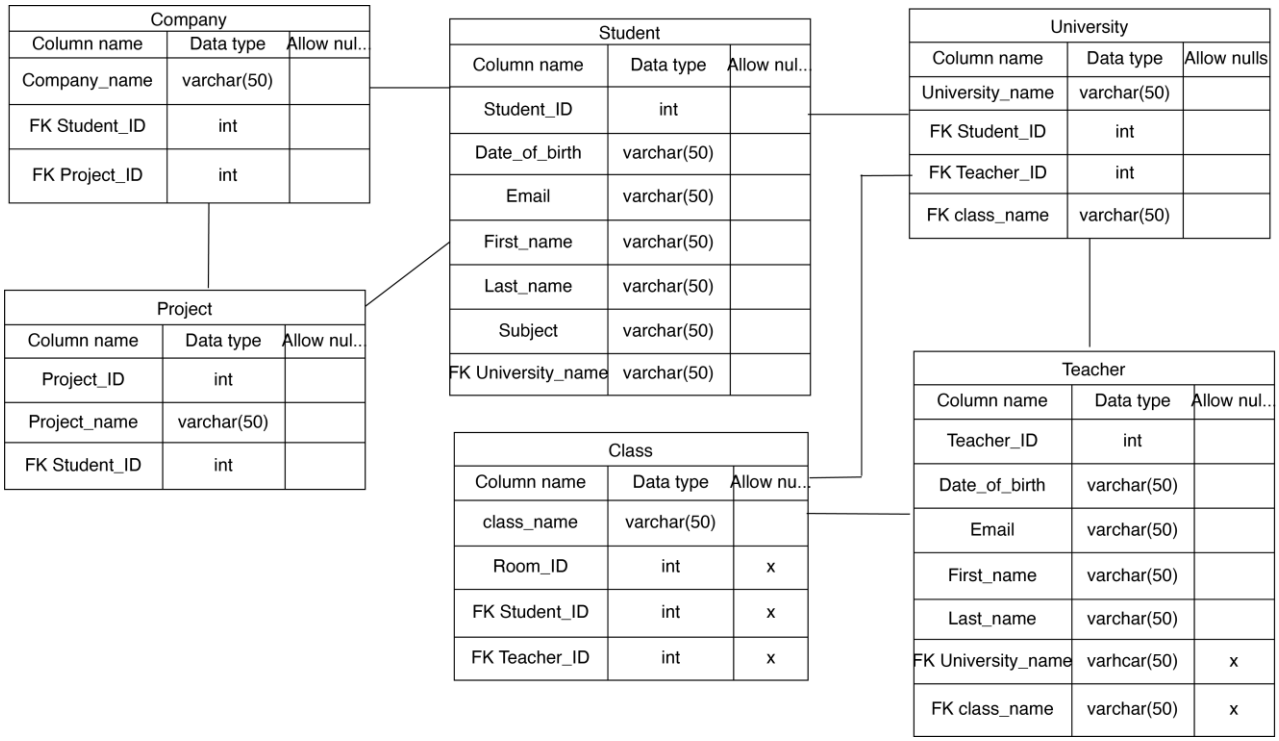

Figure 2: Relational model from the ER model

3 DATABASE IMPLEMENTATION

- Student
    - Foreign key reference to university
    - Student id, date of birth, email, first-, last name and subject cannot be null (NOT NULL)
    - Email must be unique in a way that other students can’t have the same email. (UNIQUE)
    - Student id must be unique so that other students doesn't have the same id. (UNIQUE)
    - ON DELETE CASCADE
    - ON UPDATE CASCADE
- University
    - Foreign key reference to Student, Teacher and Class
    - Name cannot be null (NOT NULL)
    - ON DELETE CASCADE
    - ON UPDATE CASCADE
    - Student and teacher id’s need to be unique (UNIQUE) even though they’re unique in the table where they are primary keys
- Teacher
    - Foreign key reference to Class and University
    - Teacher id, date of birth, email and first- and last names cannot be null. (NOT NULL)
    - Email must be unique in a way that other students can’t have the same email. (UNIQUE)
    - Teacher id must be unique so that other teachers doesn’t have the same id. (UNIQUE)
    - ON DELETE CASCADE
    - ON UPDATE CASCADE
    - There’s a check that the teacher id is always a positive number (CHECK)
- Company
    - Foreign key reference to Student and Project
    - Name cannot be null (NOT NULL) as well as student and teacher id’s

    - ON DELETE CASCADE
    - ON UPDATE CASCADE
- Project
    - Foreign key reference to Student
    - Project id and name cannot be null (NOT NULL)
    - ON DELETE CASCADE
    - ON UPDATE CASCADE
- Class
    - Foreign key reference to Student and Teacher
    - Name cannot be null (NOT NULL)
    - Room id has to be unique so that other rooms doesn't have the same id. (UNIQUE) it also has a default value of 0000 (DEFAULT)
    - Room id can be null as well as student and teacher id's
    - ON DELETE CASCADE
    - ON UPDATE CASCADE

The database also has two indices. One is based on the students last name which is the criteria to sort the data. The information is sorted in alphabetical order. The other one is based on the project and the project name. This index sorts the projects in the ascending order. These indices can be used to find the students in the alphabetical order faster, based on the last name and to find the projects on the alphabetical order.

Python interface

This python interface is run via the 'interface.py'.

We decided to create the python interface by ourselves, so we didn't use the template from the course pages. On below there's a reasoning for all the functions that we have implemented.

- **interactiveMenu** is a helper function, that renders nice menu, where user can select among the provided items. The function supports multiple selection, and inverting selection. A dictionary is passed to the function, where the dictionary's keys are the list items, and return value is one of the keys, or in case multiple selection was enabled, a list of keys.

- **initDatabase** connects to a database or creates one if such does not exist. Will return the database object as well as a cursor object.
- **funcQueries** is a menu where user can choose from multiple example queries that are executed on the database. All functions, that start with ‘query’ in their name, are available to run.
- **funcRunTests** executes the test runner bash script found in the repository under the ‘tests’ directory. This requires bash shell from the system, so will not work on Windows.
- **searchForGivenStudent** is a help function that is used to check if a specific student exists in the database based on the given id or name. This function returns the student id if there’s the student found and none if it’s not found from the database.
- **funcUpdateStudent** is a function to update information of the student. The function works so that the user can decide what information of 4 (first-, last name, birthdate, email and/or major) they want to modify. This function uses the ‘searchForGivenStudent’ function to check the student exists on the database.
- **queryStudentClasses** is a function that executes sql query. This function gives the user opportunity to list a class(es) of a specific student, or all the students and their classes. This function uses the ‘searchForGivenStudent’ to check if the given student even exists in the database.
- **queryStudentWorkProject** is a function that executes sql query. The aim is to list the specific student, its company and what project(s) they attend to and if wanted list all the students that work and their projects. This function uses the ‘funcUpdateStudent’ to check whether a given student exists in the database. It also checks whether a student is in the database has a job. There’s also option to print all the information of all of the students that work and their projects.
- **queryGetTeachers** is function that executes sql query. The aim is that the user choses a university from the list and then all the teachers from that university are listed.
- **queryStudentOrTeacherInfo** is a function that executes sql query. The aim is to print information of either student or teacher. First it asks the user to give either T or S to represent if they want to print information of a teacher or a student. Then user needs to give the last name of the person they want to search. Then the function does the quey and prints the all the available information.

- **queryContactInfo** is a function that executes sql query. The aim is to print all the contact info of all of either students or teachers. First it asks the user to give either T or S to represent if they want to know the contact information of students or of teachers. Then the function does the query and prints the contact information. The contact information in this case is first- and last name and email.
- **funcSearchStudent** is a function that allows user to search data from the 'Student' table. First it asks the last name of the student to be searched. Then it executes querie to find the information. Then the information is printed for user to see.
- **funcDeleteStudent** is a function that allows user to delete one student's all information from the table. So, the function first asks the id of the student to be deleted from the table and then executes a query to delete it. In the end of the function there's database commit so the deletion is saved.
- **funcInsertStudent** is a function that allows user to insert new student and its information into the 'Student' table. All the information is asked from the user and then it's inserted into the table. In the end is the database commit to save the insert to the database.

4 DISCUSSION

Based on the idea of the database we had very limited area to create the indices. We decided to go on with the alphabetical sorting as it helps to find the students/projects easier. Other way would've been to sort the students or teacher by their id but as the number doesn't really tell anything, the alphabetical order made even more sense.

We thought that on the function 'funcUpdateStudent' the modifying of the student table is limited only to the 4 things, which are first- and last name, birthdate and email, is reasonable. This is because we don't want that student or teacher is able to modify, for example, the id's as they're important and they need to stay the same.

Our included testing suite is packed with quite primitive tests. No comprehensive tests are included, but they have been more like a help for us to check if there are any errors regarding creating the database, syntax errors and such are captured by the tests.