327 Final Course Project Report: School Management System

Team Members: Emily Wilt, Amani Desormeaux, Nick Underwood,

MD Farhan Ahmed, Harrison Kang, Hayden Smedley

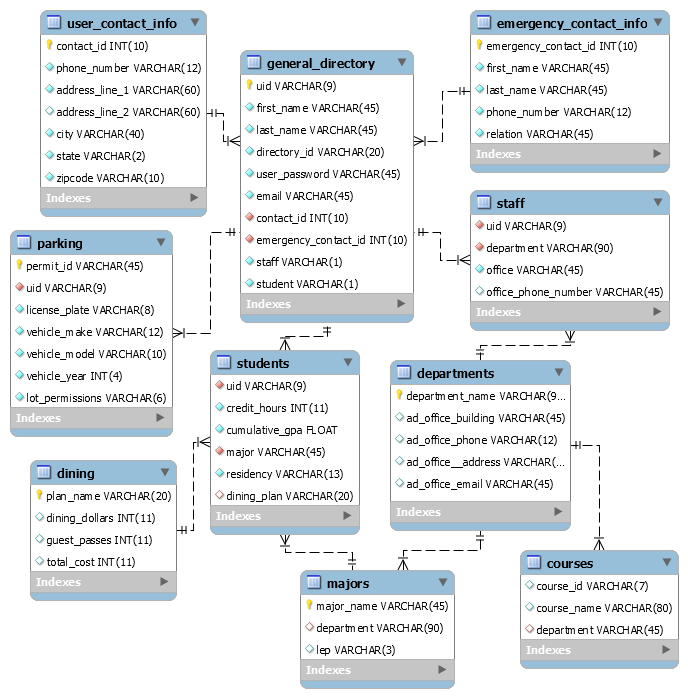
Introduction:

For our database modeling project, we embarked upon the task of creating a school management system, which, we hoped, would streamline the coordination, control, analysis, and visualization of a varied range of information about the academic programs, and individuals here at the University of Maryland. It has been our goal to create a comprehensive database in an efficient manner in which to store and interact with this information, in order to provide an easy, and excellent, experience for every user.

Throughout the process of this project, creating the overall idea, specifying necessary entities and attributes, and receiving feedback from our peers, we have fine-tuned our scope, in order to maintain an academic theme; focusing on those involved in academia at the University of Maryland, as well as administrative information pertaining to them. In order to maintain this theme and serve our desired purpose, we refined our database to contain tables about the staff, students, primary and emergency contact information, parking, dining, courses, departments, and major academic programs.

Database Description:

Logical Design: We had some design goals in mind as we moved forward. A good database system needs performance, dependability, maintenance, and meeting user criteria. We also wanted to create the logical database design as the structure of the physical database that we discussed as a team as well as the queries that it will respond to. Usually, a logical design is reinforced from the system's objects and their relationships. Furthermore, a database system which has many objects will have relationships between them. These objects can range from a simple object (the most basic) to an aggregation object (like SUM, COUNT, AVERAGE, etc.) which represents a group of entities. Considering this, we used join clause between students and major table. There were relationships that were as simple as a binary (one-to-one) relationship or as complex as a many-to-many relationship. Like the student's table and the major's table (one-to-one) as well as -general directory table and the emergency contact info + staff table ( one to many or many to many). For students table which has a direct relationship with the general directory as well as all other tables except the staff table.



Changes from original design:

Being students, we chose to have an academic-oriented project, which led us to the idea of a school management system. This is was an area that we felt very comfortable in, being able to attain enough data to proceed with the project, while also being able to create a quite comprehensive project. Our initial proposal of how we would compose our database was centered around two main tables; students and staff. After participating in the peer-review process, we realized that it would be more efficient to have the general directory be the main linking join table in our database. It was also brought to our attention that staff members could also be students, and vice versa, which led us to create “staff” and “student” columns in the general directory table to indicate whether an individual was a staff member, a student, or both. Although after making the tables, unfortunately, our database wasn’t running. Once we went over again, it was just switching up some tables at the end. Like students should be inserted at last as it refers to other departments, tables. Some of the foreign key references that we missed. After fixing this small misorder, our code ran smoothly.

Lessons Learned:

This project has taught us many lessons, ranging from working in a team to the complexities of coding in SQL. When working as a team, we would have ideally wanted to use a common server that everyone has access to and one that can be worked on simultaneously in order to disperse the workload amongst the group. Due to that option not being easily accessible to us, we decided to send back and forth the script via email which had some problems in of itself, the main problem is losing track of the latest version. But once everyone was able to meet in person, we established which version to use and everyone was back on the same page. The lesson being learned here is that online communication can sometimes be troubling but it is most effective when a team meets in person for communication to be fast and easily understandable.

Up until this final project, we only had some experience writing queries and creating a table or two so to come together to create an entire database was a step forward in our understanding of SQL. During this time we learned that a simple missing comma can cause our database script to fail and not run, giving the members a headache trying to figure out where the error is. But this kind of life lesson goes with any coding language where you realize the importance of syntax because one missing comma or semicolon can cause everything to fail.

Not only this project tested us on patience but also it tested us on attention to detail. For a very small misorder of tables, our database wasn’t responding at all. It could be by default the SQL was taking the input but we have figured it out by testing the order at the end. It taught us how to work technical project as a team. There were times when it was frustrating as we created the database and put information manually. We collaborated really well in every step of the project. Some of the members put extra effort to always make sure what the other member's task was and how to get it done.

Potential Future Work:

There are several things that we could do in the future in order to further improve this database. One of the biggest and simplest would be to just add more data. This is both in adding more records to the various tables that we already have but also adding entirely new tables to the database.

More data is always better because it means there are more entries to work with. Rather than having a database with tables composed of around 20-30 entries, we could have one that accurately represents a university being composed of thousands of students and staff/faculty. More data for majors and courses would also push our database in the direction to be representative of a university. This, however, requires an enormous amount of time but it is a task that could be accomplished by a team of people.

We previously thought up about additional tables that could be added, such as dormitory/housing information. Adding tables that further define the different aspects of the university and can be connected to other tables would be beneficial to the database as it gives us other types of data to work with and other means to characterize the students and staff of the university. These other tables would also open up the potential for other queries/views that can be created for the database. For example: If a student was found to be infected with a viral disease, we could use the database to find other students and staff that could have potentially come into contact with them be it from a dorm or from class, and screen them to make sure they are disease free.

We tried to build a database where we can simply have main information about the student and staff body of the school so that in case of an emergency, it would be easy to pull up the information that anyone needs. Such as the incident of adenovirus where the medical officers and health inspectors needed general information and in that case our database will serve the purpose. It surely won’t overwhelm you with the unnecessary information that you might not need in an emergency situation. We would love to make our scope bigger as well as create more tables that will have students particular information but at this time it wasn’t a necessary thing to do.