CIS 560 Group 7

**Hospital Logistics Information Software Development Report**

**Project Summary:**

We designed an application for a hospital to provide logistical information for doctors, hospital administrators, medical researchers, and anyone else who might be interested in the logistical details of the hospital for legitimate purposes. This application will include information about the patients, about the doctors, about the condition(s) of the patient, and about the treatment plan(s) the doctor has prescribed. Uses of this application will be to find basic doctor and patient information, link patients based on their doctor, their treatment, their condition, etc., find a specific patient or a list of rooms a doctor might be visiting his patients at, and other such connections between the patients, the doctors, the conditions, and the treatments.

**Technical Details:**

We used Azure Data Studio for our SQL database development. We created data files that included the beginning information to populate our database; most of this data is randomly generated and technically useless from a medical perspective. However, we wanted to have a significant amount of data included so that we could learn how to work with databases of significant size. We were sure to include realistic data in our database for testing purposes so we could ensure the relationships and queries all worked as intended. The relations created are: ContactInfo (to store each person’s name, contact information, and address), Doctor, Patient, PatientStay (the phrase we used to symbolize how long a patient was at the hospital, who their doctor was, what condition(s) they had, and the treatment(s) used), Condition (a list of all conditions the hospital has in its records), Treatment (to store all of the treatments the hospital has used in its records), and linking tables between PatientStay and Doctor, Treatment, and Condition used to eliminate the multiple links between these tables we would have otherwise had. All tables ended up supporting SELECT, INSERT, and UPDATE features, but excluded hard DELETE functionality in the efforts of preserving records, a focus of the medical field. Instead, each table included a final column, IsRemoved, that enables us to “soft delete” an entry should we wish it to not show up on the final search results.

We then used C# and Windows Forms for the application and GUI development. We wanted to create something that was simple and easy to use from an application standpoint because the anticipated audience may not be technically proficient to handle a more complex system. The overall design was essentially split into two halves. The first is an adding side in which users can add patients, doctors, contacts, patient-stays, conditions, and treatments. Each individual item that can be added to our system was designed to have its own page because the necessary inputs varied greatly among the different adding options. Secondly, a reading side that would allow the user to input into and see the results of the queries we designed. This side was designed to arrange the queries based on the input data, and then provide options to the user as to what exactly they wanted to see out of our hospital system based on that input. For example, a user could input a patient’s first and last name and birthday and then return the condition(s) that patient had, the treatment(s) the patient received, the doctor(s) involved in their hospital stay, their emergency and their own contact information, or the history of their stays in the hospital. The design of the system was created so as to minimize navigation as a user reads the data to streamline the process for many queries in one sitting.

**A close up of text on a white background

Description automatically generatedDatabase Design:**

**A screenshot of a cell phone

Description automatically generatedSystem Design:**

**System Features and Usage:**

As the user opens the application, the “homepage” opens up and guides the user to choose whether they want to add an item to the system or view the information already stored.

![A screenshot of a cell phone

Description automatically generated]()

![A screenshot of a cell phone

Description automatically generated]()Moving to the Update/Add to Database page displays the following page:

As discussed earlier, each of these adds required enough different information that it became difficult to design a single insertion form that could provide easy functionality to add each type of object being added to the database, so we opted to design the form to give the user ease of access as they try to add each user. So now, adding/updating a patient gives the following page (populated with data):

![A screenshot of a cell phone

Description automatically generated]()

Once the “Add Patient to Database” button is clicked, the system attempts a merge that is accessed via a stored procedure. If the merge is successful, a message box displays a simple “Data added” message, and if it was not, it displays the error that caused the issue so the developer can find and eliminate the error, or contact IT support and they can assist with the debugging (in the imaginary scenario). Now going back to the homepage and going to the search page to check the results leads to the following forms:

![A screenshot of a cell phone

Description automatically generated]()

![A screenshot of a cell phone

Description automatically generated]()

As you can see, the patient is successfully added to the database and the display feature shows the additional information for that patient. The birthdate requirement in the search feature is a requirement that ensures that one patient is selected based on the odds that it is incredibly unlikely that multiple patients share both a first and last name and a birthday.

Navigating back to the Update page, let us now add a patient stay for our example patient, Marco Polo:

![A screenshot of a cell phone

Description automatically generated]()

The Add Patient Stay form, like the Add Patient form and all of the other add forms, displays the same “Data added” message on a successful merge and the error if the merge failed. One flaw of our current design is that to add a patient stay, one must first know the patient’s ID, the doctor’s ID, as well as the condition and treatment ID numbers. We would have liked to design the system to allow a person to input names and use the search queries to find these ID numbers so that the user form is able to take those same search criteria, but due to a shortage of time, we were unable to make that change.

Now viewing the patient stay in the search part of the system:

![A screenshot of a cell phone

Description automatically generated]()

As you can see from the search results, errors in the system can result in unusual display results. As we tried to add this specific patient stay to the database, a typo in the stored procedure for adding a patient stay resulted in errors adding to the linking tables, but the actual PatientStay table merge was successful every time, and so each attempt to add the patient stay resulted in a new row in the PatientStay table, but not in the linking tables. This bug is due to the stored procedure adding a patient stay first merging to the PatientStay table, then merging to each of the linking tables, along with the DateTimePicker in the GUI giving the system’s current time if not selected, which it was not in any case. Although this bug is minor and the effects of it are not that difficult to interpret, the overall numbers of some of the other search functions might be affected, so this is a bug that would likely be patched out rather quickly if this were a real application. However, due to time and labor constraints, we did not get to fix it for this assignment.

Using the search function to display one other search, displaying patients by doctor, yields the following forms:

![A screenshot of a cell phone

Description automatically generated]()

**Report Queries:**

Our system included 4 report-style queries, as listed here:

1. A query to find all patients’ history and doctor’s information of patients who have a specific condition as well as year-to-date totals for the condition, partitioned by year, useful for disease tracking over time or potentially for hospital improvements, such as if a specific condition is prominent in the hospital and there is a unique piece of equipment that would help with that, the query could provide info to make a decision about that improvement. The user would input a condition and the program would return all patients who had that condition by ID number, admittance date, release date, and their doctor’s name and field of study.
2. A query to list all of the conditions and their treatments as well as the length of the patient’s stay that a doctor has treated based on the doctor’s information. This search could be used to show qualifications of a doctor or be useful in determining what promotions are in order. The user would input a doctor’s name and the program would return the condition, treatment, and the patient’s length of stay.
3. A query to list all patients with a specific condition by doctor, which could be used to assess a potential outbreak, in medical research to find the spread of a certain condition (i.e. how widespread is heart disease among the patients of this hospital), or by clinical trials to find potential entrants to their trial. The user will input a condition and the program will provide a patient ID number, room number, phone number, and doctor name.
4. A query to find the most common treatments prescribed for a given condition by doctor for all doctors in the hospital. This could be used to assess if doctors need training on new treatments, if they are being risk-prone or risk-averse, or other tests of care of that manner. The user would input a doctor’s name and a condition’s name and the program would return the most common treatment for every patient who had that doctor, or if no condition is provided, the treatments for all conditions the doctor has treated.

**Summary and Discussion:**

Compared to what we sought to do in our original project proposal, our final application reached most of the design goals we set. In our original proposal, we wanted to include a password protection to the application as a means of simulating a hospital’s need to keep medical records confidential, and we did not add that functionality to our application, but we otherwise included everything else that we wanted to.

We had to make a few changes from our original design in our database design, starting with adding linking tables. Our original design had just the patient, doctor, condition, and treatment tables, and each patient had many potential links to condition and treatment. We had to add a new table to include the data for each time a patient is in the hospital, and then add linking tables between this new PatientStay table and the Doctor, Condition, and Treatment tables to eliminate the multiple connections between the same two tables. Also, in our initial proposal, we had a linking table to link the Condition table to the Treatment table to show what treatments were supposed to be used for what conditions. However, as we thought about the development and usage of this application, that use seemed to be something that would be used simply by beginning doctors to examine what other doctors used to treat a condition they are unsure how to treat. But examining our queries and the usage of our database system, we realized this could be done through our other queries and did not need the extra linking table, so we removed it from our design.

This project also gave us the opportunity to learn many things about SQL and databases and how to design them efficiently and effectively. Some unforeseen lessons we had to learn in the process of making this application are developing a C# application to link to SQL that is simple in nature, having to use stored procedures rather than just normal queries so that the complicated procedures could be stored in the database and easily accessed by outside applications, and filling the data for our linking tables. When designing the C# application, we were considering using XAML to create the GUI and underlying system based on our previous experience, but both on the development side and on the user side, the end result was going to be unnecessarily complicated, so we opted to use Windows Forms and make the application more simple. The stored procedures were simply a construct we did not know about in the early design process, but once we learned them and thought about it, implementing them made the application much easier and smoother. The last thing we had to notice was when filling the data into our design, we discovered that we had to actually add data to these linking tables in order to add a doctor to each PatientStay. Initially, when designing the tables, we thought that because these linking tables only had foreign keys, they would just pull the information from other tables, but we discovered that they had to be filled just like all of the other tables.

We certainly did not optimize the potential for this application, and if this were an actual business product, there are many potential bugs and features we could use to improve the application even beyond the bugs discussed previously. Adding more queries and more types of queries is a simple design feature that we could add that would allow the program to be more useful if users asked to have a way to obtain a certain type of information. We also used randomly generated data that is meaningless to the medical field (such as fake diseases that do not exist or random people from random places), and we could clean up the data in the system to include only real cases that would be interesting, but for the purposes of this report, we just wanted to display that the code functionality worked and did not think it necessary to spend the time finding and adding that much real data for an application that would not be used in the real world. We also thought that one really useful improvement would be to make it so that the application itself stores the database information somewhere rather than each having all of the data on a local database that we then had to adjust connection strings in the GUI to allow them to access the data properly. However, we were unsure as to how to do something along those lines with the tools and skills we had at our disposal, so development of that sort of thing would take much more time than we had available. We also did not actually provide any functionality to the IsRemoved property on any of the tables, so adding that to the application is a further development we could make to improve our application. The final change we could make to make the application would be to design a similar sort of organization software to hold the other non-medical staff such as janitors and cafeteria workers so that hospital management can track those staff and their responsibilities as well as the medical personnel.