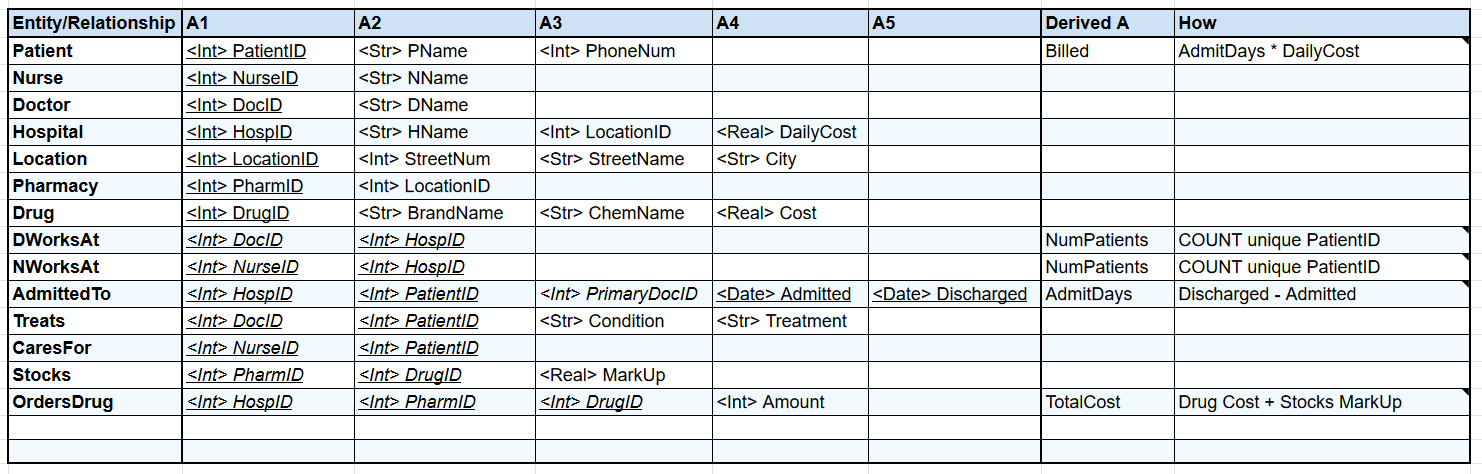
**Database Documentation**

**Database Description:** As per the project description, our database is for a system of three hospitals, each with three doctors, five nurses, and a maximum of ten patients. Hospitals also order medication from pharmacies. This is shown in our ER diagram and schema table, below.

Diagram

Description automatically generated



**Database Files and Folders:** The following outlines all the folders and files.

C3150/README.md: README file, contains all relevant information on how to use/run our database.

C3150/Documentation: All files related to documentation or descriptive information regarding the project.

C3150/Queries: All Query files including EMRSCHEMA, InsertQuery, SelectQuery, etc. (See README for instructions on how to run these). It also contains ComplexQueries which are derived attributes, and TestQueries which contain some queries for testing the constraints.

C3150/Queries/Inserts: The same as InsertQuery, but divided into separate files for each table.

C3150/TestData: csv files for using BulkInsert.

C3150/OneTouchEMR: Front-end UI made by Shieanne.

C3150/Extras: Helper files for the front-end UI made by Shieanne.

**Database Details:** As presented in our diagram, schema, and video, our solution fulfills all the requirements of the project by storing data in a 3NF normalized database. The constraints are all applied in the EMRSCHEMA, ConstraintsQuery, and TriggerQuery files, which prevent illegal data from being inserted into the database. Most constraints, keys, and triggers are intuitive, and well defined in the comments or EMRSCHEMA.sql file. Therefore, we shall explain our decisions and limitations only for the more complex portions of the project:

1. AdmittedTo has HospID, PatientID, Admitted, and Discharged as a primary key; this allows the same patient to be admitted to the hospital on numerous occasions. If the key was only HospID and PatientID, the patient would only be able to be admitted into each hospital once.

2. Treats has DocID and PatientID as a primary key; this was chosen to maintain simplicity—if a patient has multiple conditions, we thought it would be likely that a different doctor would be more proficient at treating each different issue. As you can see in our sample data, the Illness and Treatment are very general conditions (as we are not medical experts) so this simplicity functions well in our database. However, we do acknowledge that in a more complex database, a more detailed approach may be preferred. For instance, Illness might be a separate entity which is categorized by ConditionType (e.g. a variety of different heart conditions would be categorized as ‘Cardiological’). A similar approach may be necessary for Treatment, including some way to link Treatment with Drug. If we had more time, we would have loved to model this for you.

3. AdmittedTo has a TotalCost derived attribute which is calculated by multiplying the number of Days that a patient was admitted to the hospital by that hospital’s DailyCost; this was also chosen to maintain simplicity. We understand that different patients may need specialized testing or medication as a part of their stay/treatment at the hospital, which may contribute to a greater bill. However, much like the previous limitation, this complex relationship is not only outside of the scope of our project, but difficult for us as we do not have much knowledge about specific conditions, treatments, and costs related to hospital equipment/drugs. Likewise, if we had more time to research and implement this relationship, it would be a very interesting feature that we would love to add.

4. Drug has a Cost attribute, and Stocks has a MarkUp attribute; when a Hospital OrdersDrug from a Pharmacy, the cost is computed by multiplying the drug Amount \* Cost \* MarkUp, where MarkUp is a percentage. This allows Pharmacy to increase or decrease prices for certain drugs (e.g. if a drug is on sale), and a more interesting method of ordering drugs, where a Hospital can compare prices across pharmacies to choose the cheapest one.

5. Treats and CaresFor do not have an associated date; this is to maintain simplicity in the database yet again. In a real hospital, if one medical professional gets busy, they may ask another to help them with a task or patient momentarily—in such cases, the assistant would not be inserted into a database. For example, if a hospital calls a Code Blue for a heart attack, all available nurses and doctors would rush to help, and nobody would be inserting their data into a database because they would instead be focused on saving the patient. Thus, we felt that the Treats and CaresFor relationships would be most useful as a simple reference, mainly for non-emergency admitted patients. For example, if a Doctor or Nurse needs more information about a Patient’s Treatment, they can use these relationships to find which medical professional to contact. The limitation of this approach is that once a patient is Discharged from the Hospital, the relationships for that patient would have to be removed, to allow for their Doctors and Nurses to care for other patients. In a more complex database, it may be better to link the Doctors and Nurses to the Patient’s admission, which can then be stored in the Patient’s medical history.

6. AdmittedTo, Treats, and CaresFor do not check that the patient is admitted to the same hospital as the medical professional specified in insertion. For instance, Doctor with DocIDNo = 101 works at HospIDNo = 1; however, it would be possible for a patient admitted to a different hospital to have a PrimaryDocID = 101, or for this patient to be associated with DocID = 101 in Treats. This special case can either be prevented using the front-end UI or the back-end SQL (i.e. by using a Trigger which rejects insertion if the IDNo does not match staff at the same Hospital). Unfortunately, we also did not have time to implement this functionality on either end.

7. TR\_Admit\_Insert trigger is a complex one: Nonika wrote this trigger in two parts, and provided comments in the SQL file to further clarify the purpose. The first part COUNTs all the patients in the same Hospital who are Admitted during the same time period as the potential insertion. If this COUNT is less than or equal to 10 (the maximum hospital capacity), then this case passes. The second part checks that the same patient is not AdmittedTo any hospital at the same time as this would not make sense. If COUNT is equal to 0, then no conflicts were found, and the case passes. If both of these test cases pass, then the Patient can be inserted into the AdmittedTo table.

a) Note: This trigger still allows for a same-day hospital transfer; if you look at the first block in the TestQueries file, Patient 1 is Discharged from Hospital 1 on April 1st, and the only successful insertion of this Patient Admits them to Hospital 2 on April 1st to April 2nd. This same test case would have worked if they were admitted much later (or earlier) than the existing admission, but I chose these dates specifically to demonstrate a Hospital transfer.

b) This trigger also has a limitation: although we populated the database with sample values, we assume that it would realistically be used for future admissions. That is, Patients are only inserted into the database on their Admitted date, records are updated on their Discharged date. Thus, a case where a patient is inserted into AdmittedTo for days in the past is not considered.

**Our Experience**

Shieanne and Nonika both really enjoyed this project. We got the opportunity to research, learn interesting queries, and try some new front-end UI implementations. Although we did not have enough time to implement everything we would have liked to implement, we are both incredibly proud of our work and our resulting database, especially considering the limited time we had. We both agreed that this project would be interesting to continue playing around with for our own practice, and ended up discussing some interesting features that we might want to implement in the future!