CSCI 4707
Team 1
Test 4 - Synopsis

To efficiently test our database, we have decided to utilize data that we have added into the tables in the earlier test questions and reused them when the criteria are met as we continued our tests. For test 1 and 2, to test our database around Patients, we have decided to add Patients as well as a Parent and Guardian. We observed that our tables lack the criteria of having Patients who are minor to require a Parent or Guardian to be accepted as a Patient. As well as an issue for not requiring Parents or Guardians when Patients are over the age of 18. The solution to these two tests would be a trigger within the Patient to accordingly add these Patients into either Minor or Adult according to their age. Once this is done, a foreign key for a Parent or Guardian would be necessary for a Minor to be a Patient. To verify that the adding to the respective tables works, another trigger could be checked within Minor and Adults to ensure that the patients added are valid to their respected table, and if not they could be removed by their PID.

Our system documents insurance information in the Insurance_Info column within the Patient table.While the No_insurance table keeps track of patients who do not have insurance. However, we observed that when the patient's insurance information is updated, the system does not properly update the data. A similar problem happens when we test whether the system stores the patient's old and current insurance information. We observe that when the intake clerk needs to get the information for a specific patient, he is not able to determine the dates of when the old insurance was active and what is currently active.

We found in our test 5 that our system allows multiple service providers to see a patient per visit. This situation should not happen in real life. And, in test 6, we observed that one patient can have more than one diagnosis per visit which should not happen either. One major issue here is that our system does not have an entity that keeps track of patients' visits. A possible solution to prevent these happen is to create a Visit entity which has attributes of date/time, and P_ID, SP_ID, etc, and use trigger to prevent letting a patient have more than one diagnosis per visit and ensure only one service provider to see the patient per visit. In test 7, we tested whether if someone who is not a service provider could potentially make a diagnosis for a patient. We concluded that our system's diagnosis cannot be created without a valid service provider ID, thus a diagnosis can only be made by a service provider.

We found in test 9 that our system does not properly document what intake clerk collected the insurance information and the copay. Although the attributes of copay credit_card and cash are documented in the Insurance_Request table, and that the insurance information is documented in the Patient table, we did not find the corresponding data from the intake clerk's side. We had to add the patient's insurance information in the Insurance_Request table to document what intake clerk collected corresponding to the patient.

When we tried to test whether the system allowed more than one initial assessment to be in the system at a time, we found that we were not tracking the time for each initial assessment. Therefore, we can not verify whether the system can have more than one initial assessment at a time. We need to add time attributes in Initial_assessment to prevent it. After it is fixed, we will only have one initial assessment taken at the same time.

Another concern that was brought out was the fact that we did not have urgent care in our Diagram. It also goes back to where we did not have a ‘visit’ attribute in our table. The fact that we did not have an individual table for ‘visit’ made the patients’ relationships with others somewhat more complex. For patients to have relations between any employees, a visit entity would be needed if the visit table was implemented. As we started to test out our diagram, we noticed that it would have been much more efficient if we had the ‘visit’ entity.

Nurse entity needed to be fixed as well. For the nurse entity, we noticed that the nurse's information (other than their nurse id and employee id) would not be shown in from the initial assessment. Therefore, to access the nurse name in an initial assessment, it needs to have a join condition to view the information, which is additional work. Also, the same nurse can conduct two different initial assessments for different patients. The same nurse who can be identified by their Nurse ID can be in for two different initial assessment. A single initial assessment done is unique, thus the table cannot contain more than one of each initial assessment done. Each initial assessment code and an initial assessment could only contain one nurse ID at a time thus not allowing more than one nurse to complete a single initial assessment in our system.

Moreover, the initial assessment was unable to extract a certain vital report of a session with a patient. We could fix this problem by adding an ID for each seen, so that the "initial assessment" and "seen" table would be possible to join, as well as sharing a unique ID for each visits the patient makes with the doctor.

In our system, employees had sub-entities which were hourly and salary. There are no errors occuring in our system because they were two separate entities for an employee. However, to access all the employees' data, it had to go through hourly entities and salary tables to get access which can be extra work. Instead, we figured that our system can be fixed by adding a field for payment status to accommodate the employee's classification as either an hourly or salary employee.

As mentioned before our system becomes complicated for a visit due to the missing presence of a visit entity. Doctors can request more than one test/procedure per visit. Moreover, in real life, it makes sense that doctors can request more than one test/procedure since one patient might need various tests/procedures due to his health condition. Additionally, the patient may not have a test/procedure for a visit. A diagnosis is made for every patient visiting the doctor, yet a test/procedure was not created for a diagnosis. Our probable solution is to add a foreign key from Diagnosis to Test/Procedure to avoid this problem.

Another concern was with the one to one cardinality. Our diagram had Employee and Department with one to one relation. However, when we placed a different department number to the same employee id, it was not able to put the data into the table since the primary key of the employee is an employee id and the employee cannot have a duplication of the same employee. That means only one employee exists on the employee table. However, the employees were able to have a null department in the table. Another one and only one relation occurred between initial assessment and form. Though the form table can only have the one initial assessment, inserting the different intake clerk employee id with the same assessment and PID enables the user to put the same assessment and patient id in the data. Duplication of initial assessment code would not be allowed by the cardinality. We could modify the code to have the primary key that identifies the uniqueness of the initial assessment code and PID. Therefore, we only track the duplication of the combination of the PID and initial assessment code.

In conclusion, many strategies needed to be changed to make the system connect smoothly. Our team agreed that having a 'visit' entity could have made the system simpler since there are so many entities that share data. Having such an entity would allow us to efficiently test the database much thoroughly. Another probable solution to most of our problems would be a trigger for less manual need to insert data into tables for a single scenario (examples: Patients with Minor and Adult).