William Wang

Angel Tovsen

Junsoo Choi

Dr. Shana Watters

CSCI 4707

14 December 2021

**New Haven Urgent Care Database Design Synopsis**

This report will outline and summarize the testing that was performed on the database design that our team had created for the New Haven Urgent Care system.

**Requirement 1:** Based on our testing, our design currently does not meet the requirement that all patients under the age of 18 must have a parent or guardian in the system. To test this requirement, we created a patient under the age of 18, and in doing so saw that it was successful despite no parent being attached to the patient under 18. When manually forcing the patient into a child entity without a parent, this resulted in failure as a child needs a parent associated with it. In order to have a patient automatically be forced into a child upon being under the age of 18, we would need a trigger that acts on this after checking the DOB.

**Requirement 2:** Based on our testing, our design currently does not meet the requirement that ADULT patients are guaranteed to not carry the parent id information. To test this requirement we created a patient over the age of 18, and in doing so saw that it was successful. However, the issue with this is that we had to manually force the patient to be an adult and to make our design more robust we would have to add in a trigger that when we create a patient over the age of 18 it would automatically go in and create it as an adult after checking the DOB. Otherwise, this patient could be made into a child despite having an age over 18.

**Requirement 3:** Based on our testing, our design currently does not meet the requirement that a child can have a different parent for a different visit. We tested this by creating a child attached to a parent and then we went in to update the parent id of that child. The problem here then arises because in our visit table, we are only able to track the parent/guardian of a child through the child itself, and when we go to update it the information that the visit is referring back to is overwritten with the most recent parent or guardian. A better way to have implemented this would have been to attach the parent id of each child to each visit, that way each visit would be able to track the parent id of that child during the visit. If the patient is not a child then the value would simply be NULL.

**Requirement 4:** Based on our testing, our design currently does meet the requirement that a patient can be uninsured and still have their credit card information collected at a visit. To test this, we created a patient containing valid credit card information but no insurance provider value. This patient was able to be attached to a visit successfully, which meets the requirement.

**Requirement 5:** Based on our testing, our design currently does meet the requirement that in order for an uninsured patient to exist in the system they must have credit card information on file first. We were able to test this by attempting to insert an uninsured patient into the system without their credit card information, which when we try to execute this script returns an error.

**Requirement 6:** Based on our testing, our design currently does meet the requirement that insurance records can be tied to a patient if they come in a 2nd time visit without changing the insurance record. To test this, by creating 1st and 2nd visits and inserting the same patient and insurance record, both visits successfully captured the insurance information for an insured patient.

**Requirement 7:** Based on our testing, if we assume the requirement means that both multiple doctors can’t be attached to one visit and that a visit’s assigned doctor cannot be changed or updated, our design currently does not meet the requirement that a patient can be seen by only one service provider per visit. To test this, we first attempted to create multiple entries of the same visit but associated with different doctors. This did not work, as it violated the primary key uniqueness in the visit table. This would essentially mean that only one doctor could be associated with a certain visit, but further testing led us to realize that while we can’t have multiple doctors associated at once, we could update the doctor - concluding that a visit could have had multiple doctors associated with it at different times. To fix this, if necessary, we would need a constraint that disabled updates on the doctor once a visit was created. To be thorough in our testing, we wanted to mention the update issue, but if it is not a problem that a doctor can be updated, then our design does meet the requirement that a visit can only have one doctor associated with it.

**Requirement 8:** Based on our testing, our design currently does meet the requirement that a patient can be seen by different service providers on different visits. To test this, we created two different visits with two different doctors and one patient. These visits were successfully created, meaning the requirement has been met.

**Requirement 9:** Based on our testing, our design currently does meet the requirement that a given visit can have more than one diagnosis. We were able to test this by inserting in two diagnoses into the diagnosis table and then creating two new log relationships between doctor, diagnosis, and visit which allows for the visit to have more than one diagnosis logged for it by the doctor of that visit.

**Requirement 10:** Based on our testing, our design currently does meet the requirement that every visit have at least one diagnosis. To test this, since there is no foreign key relationship between visit and diagnosis, I created a log table to connect visit and diagnosis. By creating first and second visits and inserting diagnosis information by using natural join, it proved that every visit had at least one diagnosis.

**Requirement 11:** Based on our testing, our design currently does meet the requirement that only service providers can log a diagnosis. We tested this by attempting to log a diagnosis with a valid service provider id which proved successful. We then attempted to log a diagnosis with a valid nurse employee id and this was not allowed which proves that our design is functioning correctly. This is because the log entity is looking for a foreign key from a doctor and not a nurse.

**Requirement 12:** Based on our testing, our design currently does meet the requirement that the system properly documents additional tests/procedures with the proper coding. To test this, we created multiple treatments, and queried back for them in order to see that they were stored with the correct information.

**Requirement 13:** Based on our testing, if we assume that the requirement is checking to see if the assessment for a given visit could be updated after it has already been created (which to our understanding should not be possible) then our design does not meet the requirements. We were able to test this by first creating a visit with a given assessment\_id which is attached to the nurse and then attempting to update the same visit with a different assessment\_id which is attached to a different nurse. This process is allowed to go through which then proves to be an issue. Something like this can be fixed by creating a trigger that will check to see if there is already data in the field for assessment\_id and if it sees that there is data already in the column then it should not update the value.

**Requirement 14:** Based on our testing, our design currently does meet the requirement that the system allow the patient to visit the urgent care more than one time. To prove this test, I created multiple visits and inserted the same patient\_id to each visit. Both visits successfully contained the same patient\_id information and this proves that patients can visit urgent care more than one time.

**Requirement 15:** Based on our testing, our design currently does meet the requirement that initial assessments have the nurses information associated with them. Further, it meets the requirement that a nurse’s information can be retrieved. To test this, we created a visit with a certain nurse attached to it. The nurse is attached to a visit through an assessment id, which refers to the nurse who performed the assessment. We were then able to join the visit table back with the employee table and nurse table in order to get more information.

**Requirement 16:** Based on our testing, our design currently does meet the requirement that a nurse can perform initial assessments for multiple patients on the same day and on different days. To test this, we created two visits on the same day with two different patients and the same nurse attached to both, which worked successfully - showing that a nurse can do initial assessments for different patients on the same day. Then, we created two visits on different days with two different patients and the same nurse attached to both, which worked successfully - showing that a nurse can do initial assessments for different patients on different days.

**Requirement 17:** Based on our testing, our design currently does meet the requirement that for a given patient during a given visit then we can retrieve the vital information of that patient. We were able to test this by creating a visit for a patient with all the vital information that is taken during the check in for a visit and then running an SQL query on the vital information columns for the VISIT table and having it successfully return the information that was entered in.

**Requirement 18:** Based on our test, our design currently does meet the requirement that service providers request more than one test / procedure per visit. To prove this test, I created multiple treatments and requests and inserted the same service\_provider\_id (doctor\_id) and the same visit\_id. According to the test result, it successfully created a 2 result with different treatment\_code and treatment\_name. This implies that service providers can request more than one test / procedure per visit.

**Requirement 19:** Based on our testing, our design currently does meet the requirement of the cardinality rules between a patient and visit and between a doctor and visit. We were able to test the former by first creating a visit that has the same visit id but having two different patient id and when attempting to create the second visit it will return an error because the previous visit has already been made with a visit id and the given patient. To test the other direction then we created a patient that went two different visits, thus the same patient id but entries with two different visit id’s which was allowed and should be allowed to happen since a patient can go to many different visits.

To test the second part of this requirement where we want to prove that the cardinality constraint between the doctor and a visit holds true we created a two visits with the same visit id but differing doctor ids, which when creating the second visit it would return an error message because the visit should only be able to have a single doctor assigned to it for that visit. We were then able to test the opposite direction by creating two visits that have different visit ids but the same doctor id, which ran successfully because it met the requirement that a doctor can be attached to multiple visits.

**Requirement 20:** Based on our testing, our design currently does meet the requirement that primary keys are handled properly. To test this, we attempted to insert duplicate primary key values into both the patient table and the employee table. Attempting to insert the duplicate keys failed, showing that the primary keys require uniqueness and cannot be duplicated.