Project: New Haven Urgent Care Team#12

Bret Bodnar, Nicole Pupp, Audray Rong

Test 1, which checked the requirement that all patients under the age of 18 must have a parent or guardian in the system, did not pass. The provided SQL query and test data for this case allowed us to put a child into the system but did not require the inclusion of a parent or guardian. The SQL query returned an empty table since no associated parent exists for the patient. The design should work with further refinements, like a trigger or assertion that requires a parent or guardian to be added to the system if a child is added.

Test 2, which tested the requirement that patients over the age of 18 do not have a parent or guardian, did not pass. The test data successfully inserts the ID of a patient who is over the age of 18 into the CHILD table and inserts both the patient ID and a parent ID into a GUARDIAN_RELATIONSHIP. The SQL query returns the patient's information as well as the parent ID, thus a patient over the age of 18 is able to have a parent or guardian in the system. If we updated our design to include a trigger or assertion that checks the patient's date of birth before insertion into the CHILD table, then the requirement should be upheld.

Test 3, which tested the requirement that the underage patient can have different parent/guardian for different visits, passed. By applying to different EMPLOYEE and different INURANCE_COMPANY, SQL allowed us to create different VISIT. Therefore it might be better if we create a new attribute VDATE to collect the visit day for patients as a key. And the output is the table combined with Patient_CID and Parent_ID.

Test 4, which tested if the system allowed a patient to be uninsured and collect their credit card information to be charged for the visit and any supplies used during the visit, passed. The VISITS table allows the insertion of a patient with an IC_ID (insurance company ID) to be 0 and an insurance record to be 'N/A'. It then collects the credit card information for the patient. One way to make the system more effective with this test would be to allow the IC_ID to be NULL when a patient is uninsured to make it very obvious the status of their health insurance.

Test 5, which tested if the system allowed an uninsured patient in the system without the credit card information being collected, passed. When inserting a patient into the VISITS table, although it allows for uninsured patients, using an IC_ID of 0, it does not allow the credit card information to be NULL. If a patient is insured or uninsured, this information must be included with each visit for the patient.

Test 6, which tested if the system could capture insurance information and if an insurance record could be tied to a patient when they use the same insurance over multiple visits, failed. The test data inserted a first visit successfully, then failed to insert another visit with the same patient, insurance company, and intake clerk. This is because the primary key for VISITS consists of patient ID, insurance company ID, and intake clerk ID, so using all of the same ID information generates a duplicate primary key. The SQL query retrieves the first successful visit. We could fix this issue by incorporating a time and date into the primary key for VISITS, which would prevent duplicate keys from being generated when an existing patient uses the same insurance and has the same intake clerk.

Test 7, which tested is a patient could be seen by only one service provider per visit, did not pass. A patient is able to be seen by multiple service providers depending on the number of diagnoses they receive each visit. If a patient is diagnosed with two separate conditions, each of these could be assigned a separate service provider. If we wanted to guarantee a patient would only be seen by one service provider per visit, we would need to include a trigger or assertion to make sure only one service provider is assigned to every diagnosis of the visit.

Test 8, which tested if a patient can be seen by different service providers on different visits, passed. Similar to test 7, a patient is linked to the diagnosis, which is linked to the service provider. This allows a patient to have different service providers for each visit, depending on the diagnoses they receive at each visit. If this is desired, a trigger or assertion could be used to require only one service provider assigned to one patient per visit.

Test 9, which tested if a patient can have more than one diagnosis per visit, passed. The structure of our tables does not directly connect DIAGNOSIS to VISITS; this allows multiple diagnoses to be made for each visit. If these were to be connected, the diagnoses would probably be stored in the Visit_Record, which could be written to include a trigger or assertion that only allows one diagnosis per visit if desired. Or our tables could be restructured to directly connect diagnosis to visit, which could also incorporate that constraint.

Test 10, which tested if each visit must have at least one diagnosis, did not pass. As mentioned above, since DIAGNOSED is not directly connected to VISITS, there is no constraint put into place that forces the patient to be diagnosed each visit. To change this, a trigger or assertion could be created to require a diagnosis to be added if a new record is added to the VISITS table. Or the tables could be restructured to connect DIAGNOSIS and VISITS more directly and make a constraint that the foreign key between the two cannot be NULL.

Test 11, which tested whether non-service providers could make diagnoses, passed. The test data attempts to insert two DIAGNOSED tables for a patient, one with a service provider ID

and one with a generic employee ID. The test data for a service provider diagnosis runs without error, and the corresponding SQL query retrieves the patient's data and their diagnosis. The test data for a generic employee diagnosis runs with an error, as the inserted employee ID in DIAGNOSED must belong to a service provider. The corresponding SQL query will not be able to retrieve the patient's data and their diagnosis, as the insert on the DIAGNOSED table failed.

Test 12, which tested that the system properly documents additional tests/procedures, passed. We were initially confused as to what requirement 12 was stating, but we took it to mean ‘multiple diagnoses and treatments should be documented by the system’. The test data inserts two treatments and two diagnoses for a single patient. The SQL queries successfully retrieves both treatments and both diagnoses. The system does properly document additional tests/procedures with the proper coding.

Test 13, which tested whether the system allowed for only one initial assessment, passed. We included the initial assessment(height, blood pressure, pulse, weight) as attributes of PATIENT. The test data created a patient with an initial assessment, inserted their patient ID into a visit, and then attempted to insert new assessment data for the same patient. The final patient insert fails, as that patient has already been previously created. The SQL queries both attempted to retrieve the assessment data from VISIT. This requirement is functioning as described, however, lumping initial assessment as attributes of PATIENT could lead to other problems in our system. As it is currently set up, one patient can only ever have one assessment, regardless of it being the same or a different visit. We should probably include the initial assessment as a part of VISIT, or as its own table with a foriegn key in VISIT.

Test 14, which tested whether the system allowed a patient to visit more than once, passes as long as the intake clerk ID or insurance company ID differs between visits. As described in the synopsis for test 6, the primary key for VISITS consists of patient ID, insurance company ID, and intake clerk ID. Using all of the same ID information for multiple visits will generate a duplicate primary key, causing the insert to fail. The test data included three separate visits by the same patient, where the second visit had a differing insurance company from the first, and the third visit had a different intake clerk than the first. The SQL query grabs all VISIT information with the patient ID, returning three rows of data. We could fix the failure case by incorporating a time and date into the primary key for VISITS, which would prevent duplicate keys from being generated when an existing patient uses the same insurance and has the same intake clerk.

Test 15, which tested whether we can find the nurse’s information from the patient's initial assessment, passed. The test retrieves nurse information with specific initial

assessments(Height, Blood_Pressure, Pulse and Weight) for patients, especially w5rithout Patient_ID. Then SQL queries retrieve the Nurse's ESSN, Employee_ID, EFname, EMname and ELname successfully.

Test 16, which tested whether a nurse can do initial assessments for multiple patients on the same day and on different days, passed but could be improved upon. In our current system, the initial assessment data is included as attributes directly in the PATIENT table with a corresponding N_ESSN. There is no date tied to this data, so a nurse is able to do an initial assessment for any number of patients on any number of days. To further improve this system, an additional date attribute could be added to record the date the initial assessment took place, corresponding to the patient's visit. This attribute would still allow a nurse to do an initial assessment multiple times and on different days, while also recording that date information.

Test 17, which tests whether we can find the patients vital by having the visit, passed. But this test can be improved by having an unique attribute, such as visit_id. In the current system, we can find the visit by having the key, which are patient_ID, IC_ID and IC_ESSN. Then natural join PATIENT and VISITS to join the table together, finally combined select Height, Blood_Pressure, Pulse and WEIGHT from the table. For further improvement, if we give an unique ID for each visit, we can find the vital information by using the only key, which is visit_ID.

Test 18, which tests whether a service provider can give a patient more than one test per visit, passed. We can find the test/diagnosed by having patient_ID, then select the all information by having the patient_ID. The result is the table with test/diagnosed used on patients.

Test 19, which tested two foreign keys for cardinality numbers based on the design document. The first foreign key test was between PATIENT, with a cardinality of 1, and NURSE, with a cardinality of N. This test passed as a patient could only have one patient record, associated with one nurse, but a nurse could be assigned to multiple patients. The second foreign key test was between CHILD, with a cardinality of M, and PARENT_GUARDIAN, with a cardinality of N. This test also passed as a child patient is able to be associated with multiple parent/ guardians and a parent/ guardian can be associated with multiple children.

Test 20, which tested the primary keys of two tables, passed. The test data inserted a row into PATIENT and a row into EMPLOYEE. It then attempted to insert a row into each using the repeated primary key values. This resulted in the error message: "Error Code: 1062. Duplicate entry". This shows that the system is able to prevent duplicated primary keys and maintain the uniqueness of these keys.