Dylan Ceronio

Albert Levan

Carl Shefcik

11/27/2019

**CS 157A: Database Project**

Technical Documentation

**Problem Statement:**

For our project, we are attempting to implement a web-based questionnaire for interviewing tinnitus patients for eTRT, a system that supports data transactions at hypothetical clinics specialized in tinnitus retraining therapy for the management of hearing disorders. For each visit, an initial or follow up interview about the patient’s hearing disorder will be conducted and the information will be stored in a database. The database will also track information about the patient and the number of times they have visited a clinic. We will not be implementing the medical data section which are requirements 1.11.3 and 3.13.3 because professor told us that they do not relate to the interview form functionality, which is the main purpose of this project.

**Design of the solution:**

High Level Architecture:

The database was designed in Oracle data modeler and the software was used to generate a DDL file so that we may easily have all the needed SQL statements to produce a database Schema in the Oracle database server. In the folder the schema is the project2schema.ddl file.

The high-level architecture of the database consists of four entities: Interview, Visit, Clinic, and Patient. The specifics of the database is described in the description of the implementation section, but a quick overview would be that the visit entity connects all of the other entities together. A patient has a visit at a clinic and there they perform an interview. The specific relationships are also described in the implementation section.

3NF Proof of Database:

To prove that the tables are in Third normal form we must prove that they are in first and second normal form. To be in first normal form all the attributes must be singled value. In the interview entity all the questions have their own attribute and for questions that require a bubble to be filled in or to rank several issues, like the activities prevented or affected or the ranking problems questions, there is an attribute for each bubble or ranked section. Having one attribute for those questions would not be first normal form as it could have many values. The rest of questions are yes no answers or are just a text field. For the visit entity there are no multi valued attributes. For the clinic entity the address is split into address line (street address), state, and zip. For the patient the name is split into first name and last name.

To be in Second normal form an attribute must be dependent on the entities entire UID. For the interview the set of question answers depend on the UID of the interview, id. For visit the visit number and date depend on the visit ID only. The clinic address and name depend on the clinic number. The patient's name, social, email, phone, and insurance depend on their THC number. Each entity only has a single attribute as their UID.

To be in Third normal form no non-uid attribute can depend on another non-uid attribute. In the interview no question is functionally dependent on another. In visit the date of the visit is not functionally dependent on the visit number. In clinic the address of a clinic is not functionally dependent on the name of clinic. For patient none of the attributes, except for thc (the UID), depend on each other if we assume people can have the same ssn. Since our tables are in third normal form we can say our database is normalized.

UI Design:

The front end was developed using React (a Javascript library) with Node.js (the package management utility). The traversal of the website is straightforward with unavailable functionality diabled and grayed out. The actual form is a 4 part form with multiple pages with a final success page upon completion. This makes it easier for the user to digest each respective piece of the form. The inputs for the form were from another UI library: Material-UI/core. This helps our form feel professional and modern with current web dev practices. There are also many horizontal breaks to separate subsections of the form to make it easy to fill out.

**Description of Chosen Technology:** (description of technology stack, and other IT tools used (ie for design, for teamwork)

* Oracle DB 11g - Database server we are using to store our database data
* PHP - Used to connect the front end to the database
* React.js - Used to create the front end
  + Javascript, HTML, Node.js, Material-UI
* Oracle Data Modeler - Used to model the database and generate the needed DDL file
* Oracle SQL developer - editor for oracle sql
* XAMPP (Apache) - hosts the website
* Oracle instant client - to fix errors with XAMPP
* SQL Plus - This is our oracle sql command line for oracle DB 11g
* Github - repo and version control for files and file sharing

**Description of Implementation:** (description of tables, attributes, constraints, classes, etc)

Database:

The interview is a single entity that represents the interviews that patients take at visits whether it be an initial interview or a followup interview. This is done by having an attribute for every question for both types of interviews, this is possible because both forms have very similar questions except for a few. If a patient is taking an initial interview the attributes for the follow up interview will be null. The attributes are not mandatory except for the id of the interview, which is also the Unique Identifier for the entity. The Attributes are named after the interview questions. They are also categorized by the sections of the form with TIN standing for tinnitus, ST standing for Sound Tolerance, and HL standing for Hearing loss. The questions that are only on the follow up interview have the prefix FUI. The attributes are ordered in the entity by section and go from top to bottom per section staring on the left column then the right. The interview entity is related only to the visit entity. There is a 1:1 relation with both sides being mandatory. A visit must contain a single interview and an interview must be performed at a single visit.

The clinic entity represents the clinic the patient has a visit at. The entity is made up of a name, address line 1, state, and zip. The clinic number is the unique identifier. The clinic is related to the visit entity with a 1:M relationship. A clinic may be the location of many visits and a visit must be had at a single clinic.

The patient entity represents the patient that has a visit at a clinic and participates in an interview. The entity is made up of a THC number which is the unique identifier for the entity, a social security number (SSN), and date of birth (DOB), a first name, last name, phone, email, and insurance. There is a 1:M relationship with the visit entity. A patient may schedules many visits and a visit must include a single patient.

The Visit Entity represents the patients visit at a clinic. The entity is made up of a vist\_ID which is the Unique identifier for the entity, a visit number which is the number of visits they’ve had, a date for the day of the visit. This entity also connects all the other entities so it contains foreign keys (Constraints) for the patient (thc), clinic (clinic\_number), and interview(id).

The database for testing is also supplied sample data from the sample\_data.sql file. The file consists of a single patient having a visit at a clinic, there is also a sample initial interview that the patient performed.

FrontEnd:

The front end uses a special architecture known as a Single Page Web Application. This means the web page never actually refreshes even though the url may change. All HTML elements displayed are loaded and unloaded using front end Javascript without heavy implementation from the developer thanks to React.js. This leads to simple implementation and data flow control on the front and allowed our team to create a better flow without sacrificing time for data management. The form itself is split up across four different classes and files that are built and loaded into one final file that manages the data, state, and connection to the php server.

Connection:

The php server runs on Apache and also hosts our static website that displays the form. The front end sends http request to a url on the php server where our php script resides and executes a call to the database to store the data from the front end form. After completion, it sends a confirmation back to the front end where the front end will move onto the success screen.

**Instructions:** (to deploy and run the website/application)

* Unzip the files
* Make sure node.js and npm are installed
* Run - ‘npm install’ to install the dependencies
* Run - ‘npm run-script build’ to create the build folder
* Download XAMPP (Apache) and get it setup to work
  + <https://stackoverflow.com/questions/22478387/call-to-undefined-function-oci-connect>
  + This is a link for fixing some bugs with apache
* Create oracle sql connection
* Run the ddl file to populate the connection with tables
* Run the sample\_data.sql file
* Then put all the files in the build folder into xampp/htdocs
* Create the directory api/contact
* In htdocs remove index.php
* Run the apache server
* Then on your browser go to localhost/

**Contributions:**

Dylan Ceronio:

* Oracle Data modeler design of database
* Sql file with sample data
* insert\_statements.sql
* DDL File for creating the database on the Oracle Server
* Database implementation, high level architecture, and 3nf proof sections of the report

Albert Levan:.

* Oracle Data Modeler design of database
* Created the Apache server with php
* Created HTTP post scripts in php to parse and format data from front end and query database and return results
* Set up static website hosting on Apache server
* Light front end troubleshooting

Carl Shefcik:

* Built the front end website including the form
* Built the front end connection to the php server
* Helped with creating the DDL file and troubleshooting Oracle Data modeler
* Light php troubleshooting