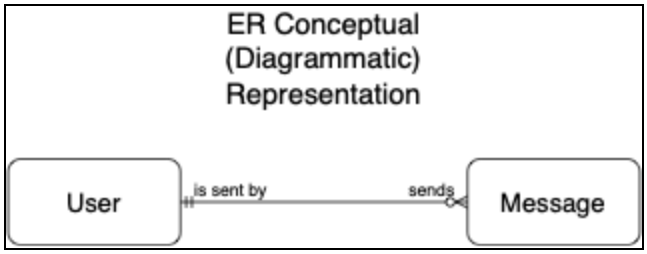
Data Modeling Assignment 4 Report

By: Anthony McIntosh and Matt Sichterman

Information about getting started, data preprocessing, importing Assignment4CompleteDump.sql to create the database, and a visual demo of the application can be seen on GitHub. The questions below give more information about the location of each of the required files.

Link to the Repository:
https://github.com/msichterman/CSCE411-Assignment4-DataAnalysisWorkflow

1. 

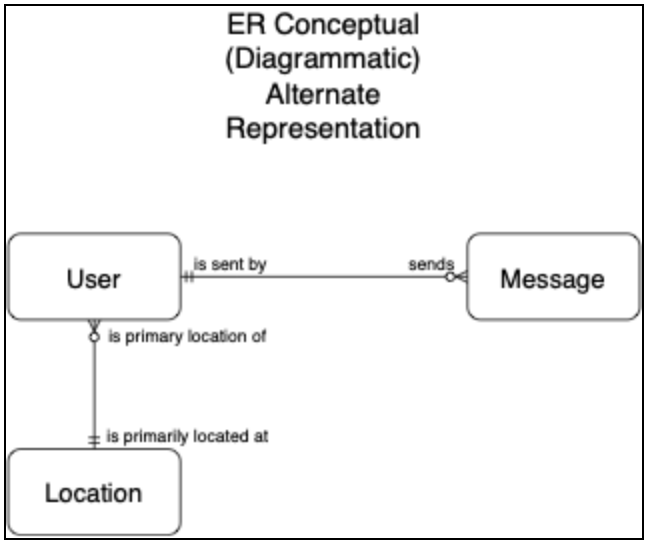
2. There seems to be two entities in this data that have attributes related to them, and those are messages and users. For example, users have attributes like their ID, Name, and Location, while messages have attributes like Sent Time, and Sent Text. Users do send messages, and messages are sent by users, however, the attributes of a message are not directly related to the user themselves. Similarly, regarding optionality and cardinality, we recognized that a User can send any number of texts, or no texts at all. However, if a message exists, it has to have been sent by exactly one person.

3. In our diagram, we can see that the text on the relation generally helps to convey comprehension of our business requirements, while the optionality and cardinality on the relationships helps to verify our design.

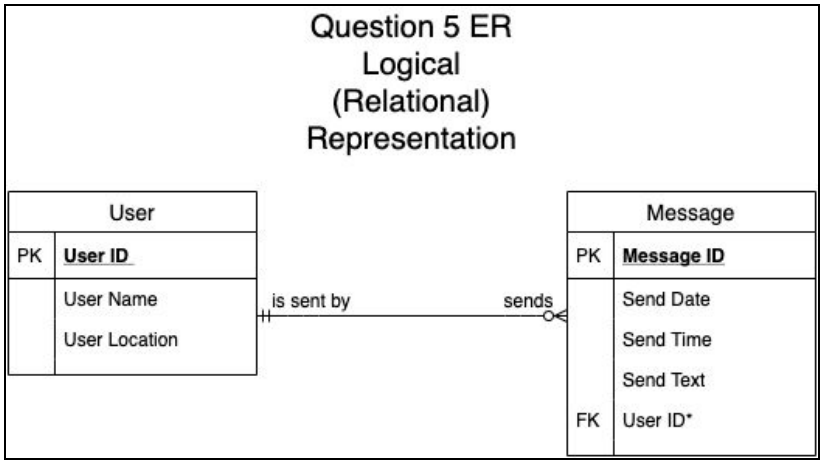
   For comprehension, we can immediately see that our system contains two primary entities, users and messages. We also can see that a message is an entity that is sent by a user, and that a user is an entity that sends a message. A stakeholder could view this and see that this design is in line with the business requirements set out by the requirements, and by their definitions of users and messages.

   For verification, we can also see the technical requirements regarding the relationship between users and messages. We can ask questions from the message perspective such as “Is it true that a message will only exist in the database if it is sent by some user? Is it

also true that a message will only ever be associated with one user?" We can also ask questions from the user perspective such as "Is it true that a user with zero sent messages will still be in the system? Is it also true that a user can send as many messages as they want?" These types of questions can be answered by the diagram, and help to verify that our design matches the business and system requirements.

4. 

4 Comparison) This alternate conceptual model above is similar to our original model. The notable difference here is that location is considered a separate entity. With the data given, we don't see this as a necessary edition, since there are no attributes related to location besides the location name. However, if more information related to location was added, such as population or weather details, separating location to its own table would prove useful. This would allow us to store these location attributes in a way that does not result in redundancy in the user table. This separation does have the benefit of allowing us to s3ee the relationship between a user and locations. We can see from this diagram that a location could possibly have no users located at it, but that a user should only have one and exactly one location associated with them.

5. 

5 Explanation) In this table, we see added information related to users and messages such as attributes, and how those attributes define the relationships. First, each entity in our conceptual model was translated to a table here (e.g. User and Message). We followed the standard rules for a one to many relationship, where the primary key from the "one" side of the relationship is used as a foreign key in the "many" side of the relationship. Every message will have a User ID associated with it to tell us which User sent that message. Since the relationship between a message and a user is mandatory, this attribute should never be blank. We can also see that multiple messages can have the same User ID as their foreign key, this is because multiple messages can belong to the same user, as seen in our relationship specification. The other information this model conveys is the attributes pertaining to each entity or table. We see that users have a User ID to uniquely identify them, a User Name, and a User Location. For messages, we see that a message has a Message ID to uniquely identify them, a Send Date and Send Time to tell when the message was sent, a Sent Text to see what the text of the message is, and a User ID to associate the message with a user. We decided to use surrogate keys for the messages, since the message time is only accurate to a minute, and it is possible for two identical messages to be sent within the same minute.

6. There was normalization performed on this data to ensure one fact per column, no hidden data, and no derived or redundant data. For instance, messages and users are not kept in the same column. Since multiple messages can be associated with the same user, if they were in the same column, there would be a large amount of redundant user data. Another consideration was made to make sure that each column only contained one fact. Specifically, we noticed that Sent Time was actually made up of both a date and a time. We separated this into two attributes, Sent Date and Sent Time, to make querying this information easier.

7. See Code - Navigate to the ‘Assignment4_Dataset_Code’ folder and in it you will see the .csv files named ‘user.csv’ and ‘message.csv’, as well as the ‘process_record.c’ which handles most of the processing logic.

8. See Code - Navigate to the ‘Assignment4_Dataset_Code’ folder and in it you will see the ‘Assignment4CompleteDump.sql’ file. See the READme.md file for more information about importing the dump file into PostgreSQL.

9. Our approach to physical modeling followed five general steps. First, read the data from the binary data files. Next, create two .csv files to represent the User and Message tables with their respective column heading, Thirdly, write the corresponding data to the .csv files. Then create the tables in the PostgreSQL database, and finally import the .csv files into the tables to populate the database. For step one, we were given code written in the C language that had two functionalities; read record and process record. Read record was able to read a specific binary file and print the results, whereas the process record code was able to read a range of binary files and print the results of all of the files in that range. We extended the process record functionality to write to .csv files representing each table that we specified in our logical modeling step. The code loops through each record and writes data to corresponding columns in the .csv files. This allowed us to model and organize the entirety of the data. From there, we created the tables in our database, and then imported the .csv files to populate each table, respectively. This gave our PostgreSQL database all that we needed in order to complete the required database queries in the next step.

10. See Code - Once the database dump is imported into a database, open up a terminal session in the root directory, install the node packages using the ‘npm install’ command and then run the application using ‘npm run start’. This will start the application on port 3000, and you can see the app running by opening up a web browser and going to ‘http://localhost:3000/’. Then click on the various buttons to make the corresponding queries to the database. The processing time for each query can be seen in the developer console.

11. Results from the queries:
    a. Find all users of Nebraska
        i. Total Number of User Records: 37
        ii. Processing Time: 18.003173828125ms
    b. Find all users who sent messages between 8am-9am
        i. Total Number of User Records: 1519
        ii. Processing Time: 78.056884765625ms

c. Find all users who sent messages between 8am-9am from Nebraska
    i. Total Number of User Records: 28
    ii. Processing Time: 53.52099609375ms
d. Find the user who sent the maximum number of messages between 8am-9am from Nebraska
    i. User ID: 985
    ii. Processing Time: 53.1640625ms