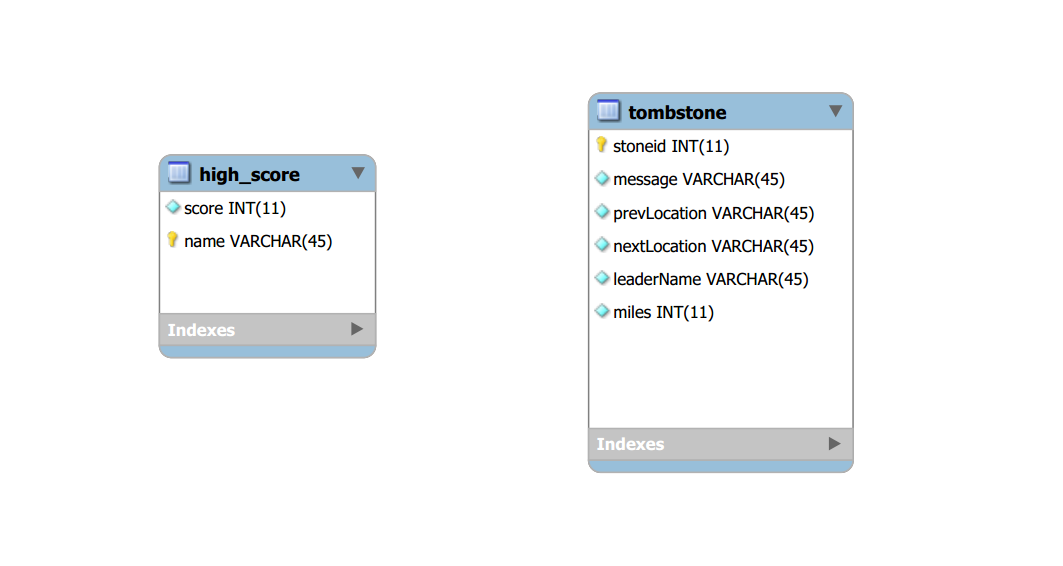
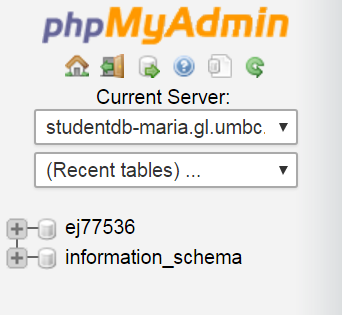
The database is setup to handle, access and store for scores and tombstones. The breakdown as follow:

The created table high\_score is to store and access the score and the name of player when he or she has finished the game. In this case, the name will be the primary key to differentiate the score that belongs to a particular person. We were intended to create an ID field for it to hold the primary key, but there was no need for it. The reason is that there is not a whole lot to deal with in term of database itself.

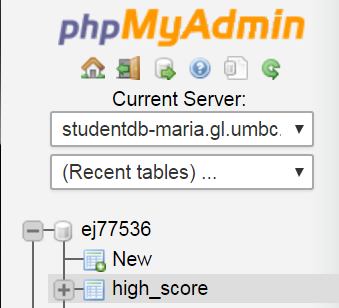
The second table tombstone is to store and access a particular message when a player die in a game and that is supposed to be written onto the tombstone. Also, the locations are there to keep track of players who have died at which spot in the game. However, to actually determine the message or the tombstone that is belonging to whom, the name of the leader of the group will be stored to display. Then again, the miles are there to see how far players have progressed through the game.

The next part of the database setup is the actual creation of the tables.

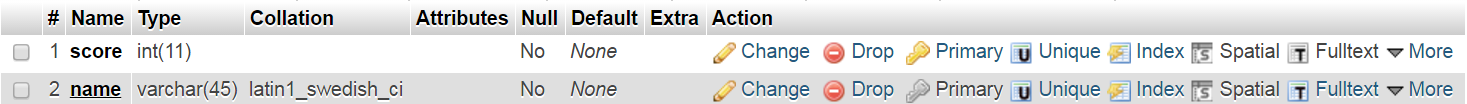
By accessing the page (https://mysql-admin.umbc.edu) and log in using the school ID. Once that step is done. It will only allow each student to have one database. In this case, mine was shown as below.

In this picture, my database is ej77536 and the server, studentdb-maria.gl.umbc, that is used to connect to add, modify, delete, or update.

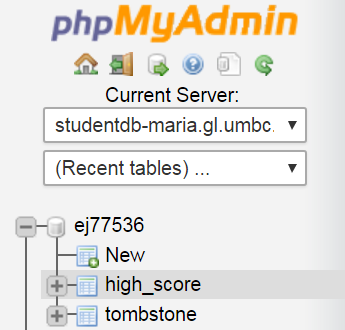
To continue further by pressing the symbol + on the left of ej77536 to expand the database for us to create new tables as mentioned at the beginning of this document. Once done that, we can press "New" and give a name of the table which is high\_score and the number of columns.



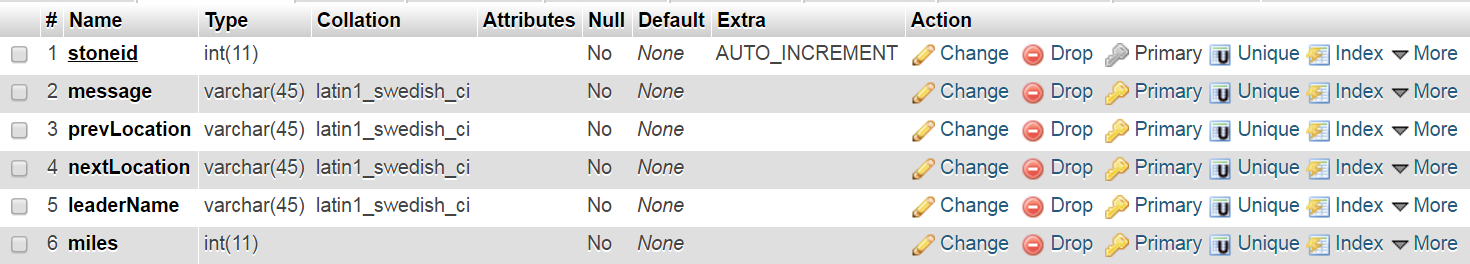
In this case, we will require two columns since one will hold the score and the other one will hold the name. For the score, it will have an INT type whereas name will have a varchar type. The name will also be set as a primary key as well. Once save, click on "Structure" and it should view as below.



After the first table is completed, we will continue on making the second table for tombstone. To continue further by pressing the symbol + on the left of ej77536 to expand the database for us to create new tables as mentioned at the beginning of this document. Once done that, we can press "New" and give a name of the table which is tombstone and the number of columns.



In this case, we will require six columns since the first one is to hold the id; the second one is to store the message that is inputted after player dies; the third and fourth ones are for locations of previous and next; the fifth one is for name of the group or leader's name, and the last one will hold the number of miles. Everyone of them will have a varchar type, except for the id and the miles fields. Also, the id field will be set as a primary key and check auto increment. Once save, click on "Structure" and it should view as below.

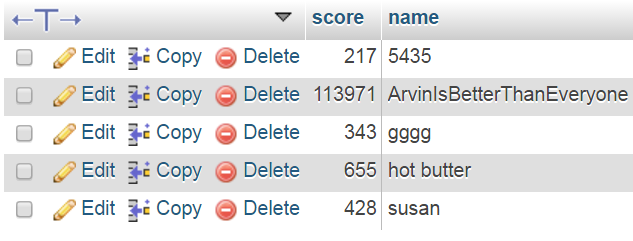


After that, it should be it for creating table. Then, we run a couple SQL queries.

To insert a score and name for high\_score table using statement:

INSERT INTO `high\_score`(`score`, `name`) VALUES ([value-1],[value-2])

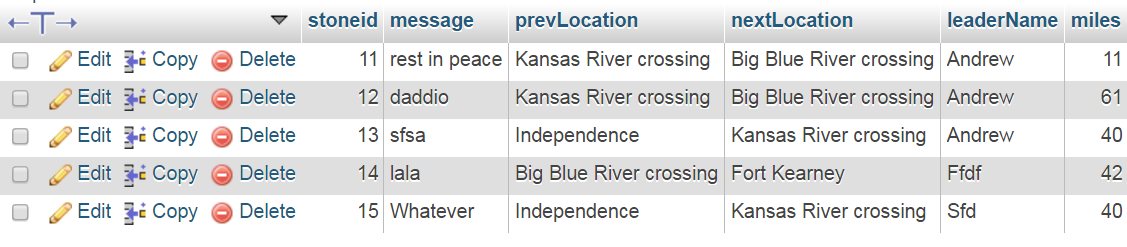
Replacing [value-1] for score and [value-2] for name and we have.



To insert a message, locations, name, and miles for tombstone table using statement:

INSERT INTO `tombstone`(`message`, `prevLocation`, `nextLocation`, `leaderName`, `miles`) VALUES (value-2],[value-3],[value-4],[value-5],[value-6])

Replacing [value-2] for message, [value-3] for previous location, [value-4] for next location, [value-5] for leader's name, and ]value-6] for miles, and we have.



The sample code using PHP on XAMP or Wampserver on the actual game to perform the insertion.

