I chose document database design because when I look at the databases it makes the most sense. I'm not going to store a lot of database only in memory. So, for what I'm doing it only requires a short-term database. There are only two elements, or relationships involved, word and user. These elements don't have anything to do with each other so a document database will suffice. There is no relationship where tables need to be drawn.

Json Structures and types:

Users: {

userId: integer

username: varchar

password: varchar

gamesWon: integer

gamesLost: integer

}

This json structure has a name of user. The user name comes from the fact that there will be user that will be playing the hangman game. The userId will be important to the user because it identifies the user. There will be a username and password for logging into the site. So, the user will need to login to the site every time, unless they have username and password saved in the browser. The username and password are also identified as having varchar type, which is a text string type of type. gamesWon consists of data used to acknowledge the number of games that have been won. gamesLost, signified by an integer type are stats that show the number of games lost. Each field in the object is important because it displays some different data.

Words: {

wordId: integer

word: varchar

timesUsed: integer

timesNotGuessed: integer

timesGuessed: integer

}

The words object is different from the user object and will never cross paths. This is why they are both considered unique objects that do not share a relationship. Starting off, the wordId is used to distinctly identify which word is being used. Without it, the object wouldn't have anything that distinctly represents it in integer form. So, the word Id distinctly represents the word object. Word is next. Word represents the distinct word being pulled by the computer to be the word that is being guessed by the

user. TimesUsed represents the amount of times the word has been used. TimesNotGuessed represents the amount of times the user guesses the word letter incorrectly. TimesGuessed represents the amount of times a user guesses per game. So, each letter of the word that they guessed correctly is accounted for here.

Stretch Features:

Timer {

timerId: integer

timer: integer

timerGo: integer

timerStop: integer

}

A timer is distinct. It separates how the game was to how it currently is. Adding a timer is a stretch feature that adds to the value of the game. Starting off, the timer ID identifies the timer. Of course timer is the working clock of the timer. TimerGo represents the action of the timer when it is in go form. TimerStop represents the timer at the point of which it stops, usually this will take place at the end of the 2 min. time frame the clock runs in between letters called.

totalScore {

totalScoreId: integer

totalScore: integer

highScore: integer

lowScore: integer

}

The totalScore calculates both the total score and the high/low scores. The score will be either high or low, depending on what the score is at the end of a game. I have yet to decide whether the game will add points from each game to make the totalScore or add them up per game. Starting off, the totalScoreId identifies the totalScore object. The totalScore adds up the total score points whether that be adding or subtracting the letters gotten wrong per each try. The highScore attribute tells whether there is a new high score. The lowScore attribute tells whether there is a low score. This will be done as a default listing of the score.