Zachary DeVita
CS 496
Assignment #3

For my API I will be modeling course data that could be utilized by a school. I will be using three entities in the API; one to hold an object for each student, a second to hold an embedded document for courses, and a third to hold another embedded document for department. The entity for *Student* will contain the student's name, their student ID, and a list of embedded documents for courses which the student is currently enrolled in. The entity for *Course* will hold the name of the course, the instructor's name, and the embedded document for department. The entity for *Department* will hold department name, and its location.

Entity	*Single Document*	*Single Document*	Embedded Document
Student	name	student_id	[course]
Course	course_nm	instructor	department
Department	dept_nm	location	

An example of the DynamoDB JSON object might look like this:

```
{
    name : "Billy-Bob",
    student_id : 123456,
    course : [{
                course_nm : "cs496"
                instructor : "Mike Judge",
                department : {
                                dept_nm : "computer science",
                                location : "Kelley Engineering Center",
                }
            },
                course_nm : "geo321"
                instructor : "Greg Graffin",
                department : {
                                dept_nm : "geology",
                                location : "Weniger",
                }
    }]
}
```

For the assignment I will be utilizing DynamoDB for my query language, and I will be using *Amazon* to host my API with their AWS service. The database will be organized into JSON objects, one for each student. There is a single identifier, or *hash*, for each student, and this is the student ID field. This student ID is a unique value for each student and allows for querying data relating to the student. The student name could be used as a secondary key, also known as a range key, but since the student IDs will be unique it is not a necessary feature.

A query on a specific student will allow the user to identify personal information about the student, and what courses the student is enrolled in. Each of the courses the student is enrolled in has further

information which is also accessible via querying the same collection due to the non-relational structure of the database. This is also true with the department information for each class. All of this information is accessible in a single transaction, and without needing an expensive join to do so. This structure is a bit redundant and requires much more memory to store the database, but, if the difference in space isn't an issue, the data queries require far less time in many cases.

A huge benefit of this structure is that data is stored in serialized arrays of pseudo-JSON objects. JSON is very easy to work with, and is commonly utilized by many other backend languages and JavaScript, making for an intuitive transition for web development. It is also convenient for storing multiple records with no unique fields in the same collection of data. For my implementation each student will have a unique ID, but I can definitely see a benefit for this functionality.

Non-relational databases are probably most advantageous for huge and complex data storing. For the purpose of scalability, the comparison is staggering. There is far greater scalability in non-relational databases which will greatly decrease the amount of de-normalizing your database schema.

I would say that MongoDB is the most similar in nature to DynamoDB's functionality. One of the primary differences I could find between the two is that MongoDB doesn't use keys/value pairs in its objects. Instead, each item is a collection of attributes which can be used in queries. This makes MongoDB much more flexible in development, robust and simple. What MongoDB also has going for it is that it is not locked into a single platform provider. DynamoDB can only be utilized using Amazon's AWS platform.

With that said, I personally love Amazon's AWS platform, and I use it to host all my databases, relational or not. I love the features that are provided with the platform as well. I like the key/value pairs; though the programming might not be as simple, it is a structure that I have grown accustomed to using JavaScript over the years. Not that this would probably be an issue for a simple classroom built API, but maintaining and scaling with DynamoDB is far superior, and there are built-in features for cluster performance monitoring that MongoDB does not have. With DynamoDB, all data is stored in fast solid-state drives as well.

I personally believe it completely depends on the project for determining which of these two query languages to use. For the project I am doing it would be probably be easier to use MongoDB, and I wouldn't run into any of the problems associated with it either, but I still had to go with DynamoDB. This is because I am fairly familiar with the AWS platform and because I believe DynamoDB is blowing up much faster than MongoDB. I am more likely to run into DynamoDB in my future and, for that, it is in my best personal interest to gain the experience using it.