Mackenzie Holznecht
Assignment 3: part 1

Describe the data you will be modeling

For this assignment you will need to have at least two types of things with at least two properties each, though something a bit more complex than this would be ideal. These things should be related in at least a one-to-many relaitonship, for example products and shipments, classes and students etc.

The data I'll be modeling is a physician's office storing patient's insurance information. Although in real life I would have to worry about being HIPAA compliant, I'm not going to worry about it for the purposes of this assignment. The patient data kind will have a required name and date of birth (ideally used in combination as a unique identifier as people can have the same name but different dates of birth), as well as additional information such as address, city, state, zip, phone number(s), an insurance ID (can be a combination of letters and numbers), etc. The insurance data kind will have a required name, an effective date, a copay amount, a provider assistance phone number, etc. I haven't exactly figured out the properties because I'm not sure how in depth I'm going to go. I'd like to start basic and get it working, and then add more properties and functionality as I go.

Come up with a plan to model the data in a non-relational database.

You should discuss how the database stores data, how you will set up the database to store your data and why you chose to set it up the way you did

The data will be modeled in Google AppEngine's Datastore, which is non-relational (my reasoning for the choice is described below). All of the logic and coding are planned to be be done with Python and the GAE framework/libraries – specifically ndb. Both patient and insurance kinds will be able to be added, edited, and deleted, and ideally queried (an add-on assuming I'm able to get everything working the way I'd like) to sort out patients with various kinds of insurances, etc.

The one-to-many relationship is that, unlike real life, patients in this system are only allowed to have one insurance (in real life people can have multiple, supplemental, and secondary insurances). This system will deal with only a primary insurance provider. It goes without saying, many people will be able to have the same kind of primary insurance.

The way I'm envisioning it right now is that the patient kind will have all of its properties, as well as a Key Property which will identify the key of the insurance kind with which they're affiliated. I didn't want to denormalize the insurance name in the patient kind because insurances often come out with different levels (and therefore different names) to the plans they offer. If one plan "splits" into two separate plans, I didn't want this to require an update to all patient kind properties with that insurance. I realize this would happen rarely, so for efficiency and scalability, it may be wise down the road to denormalize this data (a lot more reads than writes happen to the data – generally, once a patient's info is in the system, it rarely changes, but is read quite often). For this assignment purposes, my thoughts were to start basic and add complexity after. Although this may make it more difficult in the future, I think it'll better help my understanding.

I'll be able to use some of my code from last week, but I wasn't sure then what I wanted to do for this assignment, so I'll have to reconfigure some of it and change the basic names and properties around.

Mackenzie Holznecht
Assignment 3: part 1

Research one additional non-relational database.

Discuss the differences in how data is stored and what impact that would have on the way you choose to model the data.

For my research on an additional NRDB, I chose AWS's Amazon DynamoDB. It's a fairly easy comparison for a newbie like myself because they're both non-relational (duh) and are hosted by huge companies with a lot of development ideas/capital/etc. at their disposal. These resources make it "easy" for them to come up with efficient and intuitive features for a beginning developer like myself. (Side note: I realize I say efficient and intuitive which are both very subjective terms, but in regards to beginning level development, they've got HUGE resources at their disposal to make the easiest, most user-friendly platform.)

The first, and in my opinion most significant, difference of features between Google AppEngine is the use of indexing in the non-relational databases. GAE's datastore supports both single and multi property indexing making queries more time-efficient and cost-efficient. DynamoDB doesn't support indexing at all – a developer must account for all aspects of indexing on their own, making this a bit more intricate in terms of design and maintenance. As mentioned in the lectures this week, complicated queries and seeks for data are expensive, so the automation of GAE's datastore indexing helps in deciding which platform to use, not only in terms of experience as a new developer, but also in terms of maintenance of indexing in the future.

Another notable difference between the two is the ability for GAE to support transactions across entities, whereas AWS's Amazon DynamoDB does not support this. Using a posting comparison from the Piazza discussion board, another student made this table in comparison:

Amazon Web Services (AWS)	Google App Engine
DynamoDB	Datastore
Table	Entity
Items	Kind
Attributes	Properties
Values	Values

Entities are GAE datastore's way of organizing data under one parent to help with ensuring *strong* consistency during transactions. GAE still promotes eventual consistency, but with the use of Entities all stemming from a parent, it can promote strong consistency within that entity. Also, GAE supports transactions across entities – I believe a discussion board post said it's been updated to 25 entities can be updated in one transaction. In dynamoDB's platform, transactions are limited to only one table at a time.

These two differences are the main factors I've found separating GAE and AWS NRDB platforms. The automation of indexing and the ability to transact across entities is why I chose to stick with GAE. That, and the lectures are all referencing GAE so it makes it easier for me to follow and comprehend. As a beginner, the more I can conceptualize the information, without having to delve deep into the details, the better I am able to learn the material.