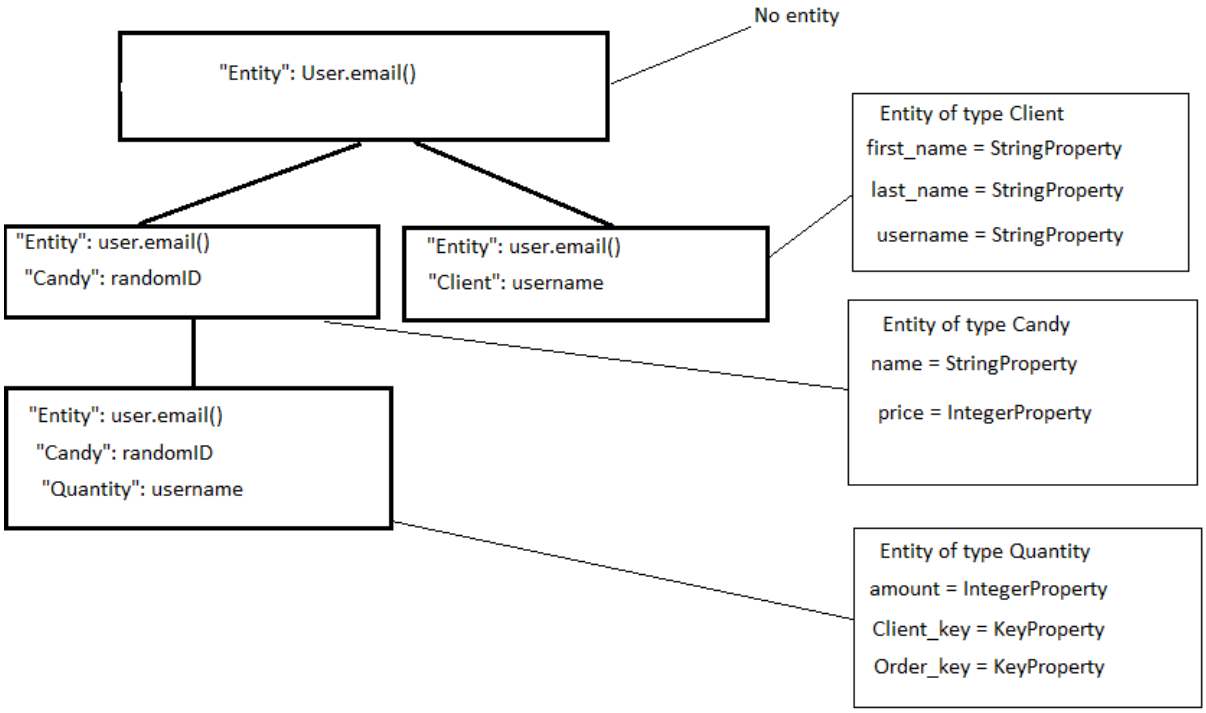
Minh Nguyen

Assignment 3 Part 1: Designing

I will be modeling a simple e-commerce web application designed for candy distribution companies to use in order to input its transaction into a database and view this information. The data store that I will be using is NDB from google app engine. There will be 3 entities involved; Client, Candy, and Quantity. The Client entity will have 3 properties; first_name, last_name, and username which are all string properties. Candy will have 2 properties which is name and price, name being a string property and price will be an integer property. Quantity will have only 3 properties; amount which is a integer property, Client_key which is a Key property, and Order_key which is also a key property. Client_key and order_key are key properties in the event that the user wants to query and view data a certain way (View how many Client's ordered a certain quantity of items, or view what items were ordered a certain quantity).

Below is a diagram of how the hierarchy of entities will be used according to the way that Google app engine's data store is set up.



When setting up this data base, the root entity here will point to a non-existent entity. In GAE, the root key is allowed to point to nothing, since it is created primarily as a naming schema for its children key's. One step below the root entity will be the User and Candy entities. Users and Orders will automatically be connected since I will be implementing user login with GAE's "users" library. This makes it so a user signs in using Oauth2 and will be displaying only your data inputs. This design choice makes it unnecessary for Candy to have a unique key to go with the User entity. The child node will be the Quantity entity. The Quantity entity is the lowest level node in the event that the user wants to perform statistical analysis of their products. As mentioned above, having the Quantity be the child node will allow users to view data depending on the user_key or order_key.

The schema is designed this way with the idea of using strong consistency. Strong consistency will allow for data to appear immediately since this is a small assignment. There will be not millions of users on the web app at once, thus the need to overcome the 1-transaction per second rule for GAE's ndb is unnecessary.

Data will be entered via an html template. Users are free to add in as many Users as they like, as well as different types of Candy's. They can then also add the Quantity for each of their Client's order's and it will be saved to that particular Client's information. There shall be a view page that displays all information necessary for the user. In the view page, it will display their client's name, orders and quantity number.

The other data store that I researched was Amazon's simpleDB for non-relational databases. It closely resembles a relational database in the sense that domains are used, which are similar to tables. SimpleDB is still a non-relational database which means that "items" represents individual objects similar to entities. Items can have attributes, which are similar to key properties in NDB, and each attribute will have a value. Items can exist in one domain, which is similar to having tables in a relational database in the sense that item's can be queried in relations to other items in the same domain. Users can have several different domains depending on their projects. Items cannot be queried across different domains however.

Queries in SimpleDB resembles that of MySQL. It's basic syntax is select * from MyDomain where item = MyItem. In my opinion, SimpleDB is more organized and easier to grasp than google's NDB. If I had to implement my candy distribution web app, I would create a domain called "candy distribution". Items would be the 3 entities I have; Client, Candy, and Quantity. But without the key property that GAE's NDB has, I would not be able to create my app exactly to specification since the

Quantity item will not have key properties in order to connect it to Clients and Candy. Here would be the reason why I would not implement my project in Amazon's simpleDB.