Vlad Predovic

CS 496

Due 10/18/2015

HW3 Part 1

For my project the two related things I will be working on are Time Study Projects (*class: TSInstance)* and Project Procedures *(class: studyProcedure)*. A Time Study Project is an overarching process that a user wants to gather data on. It will have properties such as Date Started, Name, User, and associated procedures. Additionally I added an Endfigure implementation similar to the one used in week 2's example to highlight the project the time study was working on. This set of attributes was created in mind to allow for multiple instances of the same study to exist while being able to order them and still connect them to a user for data gathering purposes. Therefore, the Date Started field is critical as it allows the user to implement multiple instances of the same study under their own User-name and the same unique key

Project Procedures are the many specific tasks that make up a time study. These procedures will be 'sub-items' of each time study project, indexed by their startTime and more importantly the specific instance of the current study iteration. The procedures will have two other properties, the procedure name and the ending time. Taking advantage of the flexibility of Non-Relational Databases allows for the sub-instances of each completed study to exist under the same parent Time study.

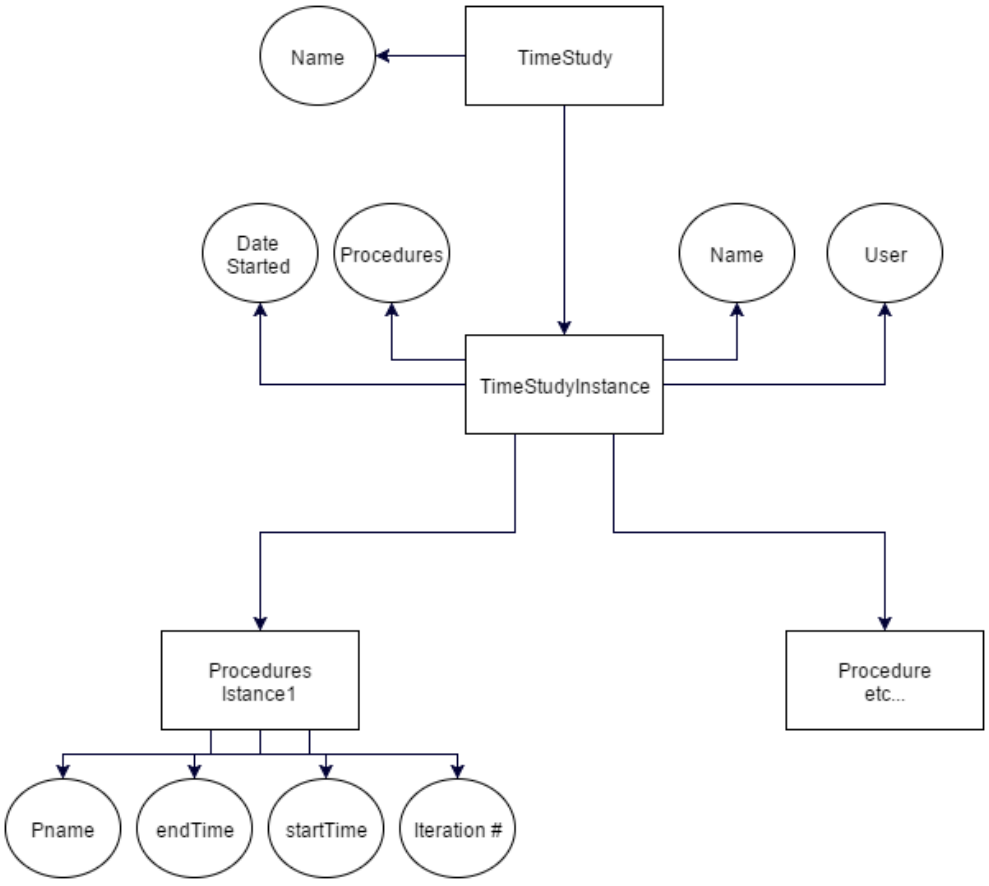
However, these sub-entities all exist under the TimeStudy parent entity as shown in the diagram below representing the Entity Group. The goal of this form of organization is to allow a user to easily gather all study results done over time on the same study efficiently and quickly regardless of the scale.

```
1   from google.appengine.ext import ndb
2   #database definitions
3   class TimeStudy
4       name = ndb.StringProperty(required = True)
5
6   class TSInstance(ndb.Model):
7       TSname = ndb.StringProperty(required = True)
8       user = ndb.StringProperty(required = True)
9       Procedures = ndb.KeyProperty(repeated = True)
10      dateStarted = ndb.DateTimeProperty(required = True)
11      endFigure = ndb.BlobKeyProperty()
12
13  class StudyProcedure(ndb.Model):
14      Pname = ndb.StringProperty(required = True)
15      startTime = ndb.TimeProperty(auto_now_add = True)
16      endTime = ndb.TimeProperty(auto_now_add = True)
17      iteration =  ndb.IntegerProperty(required = True)
18
```

In order to model the data in a non-relational database I have designed it in a way so that each project sub-entity is its own data set. Although identical procedures and projects could exist if different people are doing studies on them, these entity sets will still be portrayed as separate due to their key combination beneath the over-arching Time Study parent. Modeling the data autonomously allows for better sharding by having all the data for each separate study concentrated in one area. This would become necessary if for example this application was implemented at a large factory where people began producing thousands of studies. Additionally, since this application is study-focused, users are not really needed. A simple input for the name of the study holder is sufficient for future queries.

My implementation will utilize Google's NDB, the Datastore, since this is the platform I have been working on. The Datastore uses Entity-Groups to model their data. All data exists in a form of entity tree where entities can continually have more nodes. One of the advantages of using my model is that I only have one entity group, removing the threat of reaching the 5 entity group limit. Unfortunately, there is only one right per second which could cause issues because it is possible that multiple people will be running studies under the same entity group. Further tests are needed.

Below of a visual depiction of how my entity groups will exist autonomously within the Datastore.



I investigated MondoDB, a popular service for big data applications. MongoDB scales by adding more servers and modeling data as JSON documents. The company claims this kind of modeling system allows it to evolve effortlessly without having to resort to duplication. After reviewing how data is stored in MongoDB I have found that at least from a querying standpoint it is similar to the Datastore. In MongoDB you create ‘documents’ with your data, which gets saved as JSON data. However, it is also possible to embed documents within documents. This is where the non-relational aspect comes in by allowing you to model your data as a set of binary trees. In MongoDB data form different documents can also be connected in a way by using references that have the same value in each document. In a way this is very similar to the entity grouping method that google provides.

MongoDB has the same limitations as many other non-relational databases. For example, there is no equivalent to the SQL join. Therefore the ability to get data with somewhat relative ease depends primarily on your data model, which can vary with the direction of the process it is modeled on. Remodeling the models is painful, as you need to pull the data, make the changes, and then re-upload the data in the new format. Also, debugging in MondoDB is very difficult. Additionally, MondoDB has no support for traditional locking or complex transactions. This would become an issue in my application when multiple people are accessing and printing out data to do calculations on.

Overall, MondoDB is meant to be a lightweight application for manipulating simple yet enormous quantities of data. The syntax is similar to Google App Engine’s and were I to scale up I believe it would be a good choice, as it does not have many of the limitations that the Google App Engine does for traffic.s