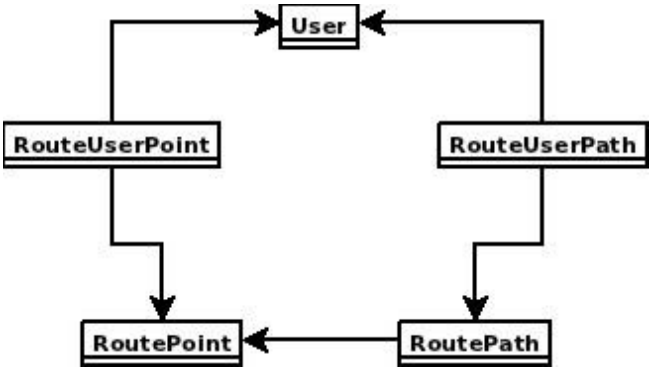
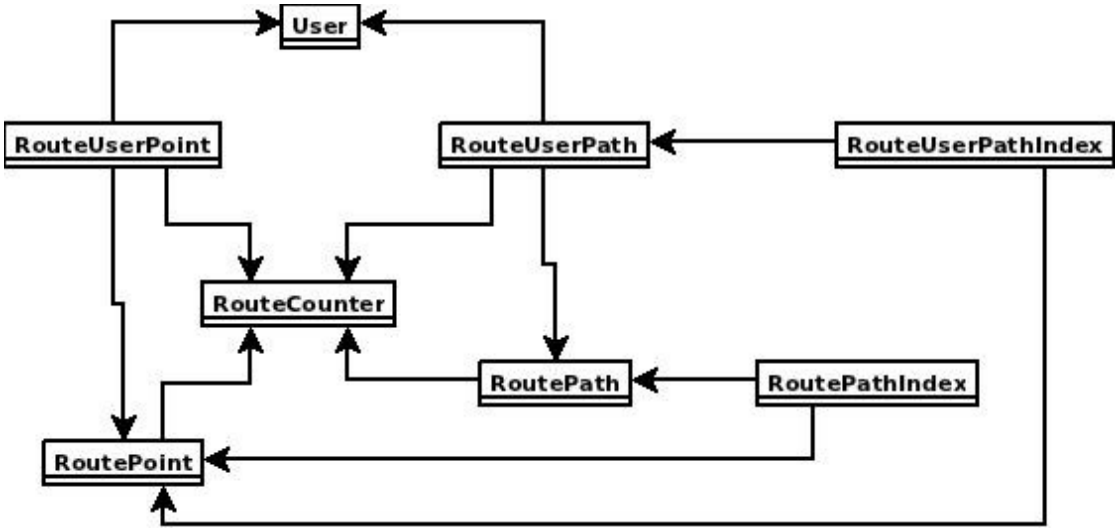
Documentación GeoRoute 0.1

Modelo 'general' para BD:



Modelo para DataStore:



¿Route...Index?¿Por qué no usar simplemente una propiedad ListProperty en RouteUserPath y RoutePath?

Utilizar listas como propiedades es lo mas sencillo, pero tienen varios pros y contras:

- pros:

  Rapidas en la escritura.

  El coste de CPU escala linealmente con el numero de elementos.

  Tienen pocos costes de espacio en disco.

- contras:

  Elevado coste de serializacion/deserializacion cada vez que se accede a la instancia.

  En cada lectura se deben crear todos los elementos.

  Si el tamaño de la lista > 100, se vuelven muy lentas

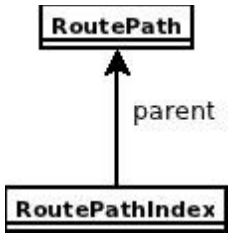
¿Cómo lo mejoramos?

Si pudieramos hacer un `SELECT a,b FROM Model`, podriamos evitar cargar la lista si no es necesaria, pero el DataStore no soporta ese tipo de consultas.

Separamos en dos entidades, asi solo nos preocupamos de la lista cuando sea realmente necesaria, ejemplico:

`RoutePath` → tendra todas las propiedades del para el Path.

`RoutePathIndex` → tendra solamente la lista de puntos que formar el Path.



Podemos utilizar mas posibilidades del DataStore como son los grupos de entidades, de esta manera, podremos modificar ambos objetos dentro de la misma transaccion (no se permite modificar varios tipos de entidades en una misma transaccion), ademas de que las entidades de un mismo grupo, son almacenadas de distinta manera para permitir un acceso mas rapido entre ellas.

Obtener las claves de paths coincidentes a un punto:

```
indexes = db.GqlQuery("SELECT __key__ FROM RoutePathIndex WHERE points = :1",
      point.key())
paths = [k.parent() for k in indexes]
```

Con esto conseguimos:

Escrituras con el mismo coste, pero lecturas ~10% mas rapidas.

Poco coste de espacio en disco, poco coste de CPU

Los Index son escalables, si llegas al limite de su ListProperty, creas mas Index con el mismo parent.

Si quisieras filtrar una consulta para obtener que Index tienen el mismo padre, puedes usar, `ancestor(`*`ancestor`*`)`

¿RouteCounter?

Aproximadamente cada entidad o grupo de entidades, puede actualizarse sobre unas 5 o 6 veces por segundo, por lo que si tuvieramos un contador que necesitase muchas actualizaciones, podriamos crear contencion y ralentizar la actualizacion de esa entidad.

Por lo que creamos contadores fragmentados, son rapidos en la lectura, ya que no tienen el resto de propiedades que si estaria en el modelo original, permitiendo mayor frecuencia de actualizacion

Los costes de lectura son minimos en el DataStore, por lo que no hay problemas al leer varios fragmentos para calcular el total, mas facilidad para utilizar memcache.

La escritura 'se parte' cada vez en una transaccion que afecta a uno solo de los fragmentos.

Enlaces:

[1] http://www.scribd.com/doc/16952419/Building-scalable-complex-apps-on-App-Engine

[2] http://code.google.com/intl/es-ES/appengine/articles/sharding_counters.html