Google app engine: Assignment 1

Wouter Vandenputte & Jorden Belis

23/11/2018

Questions

1. Motivate your solution that provides the all-or-nothing semantics when confirming quotes. For example, why did you (not) use cross-group transactions (XGT)? Did you rely on JPA transactions? What are limitations of your design? Justify your choice, assuming the following dimensions of a car rental agency service in practice: 32 registered car rental companies, and holding together more than 95 car types.

We don't use cross-group transactions because there is no need for them at this time. We rely on JPA transactions and use an EntityTransaction to garantee all-or-nothing semantics.

2. Let another, independent part of that rental agency application on Google App Engine manage data related to client profiles. These profiles –whose data model is shown in Figure 1– store data that is exclusively related to a single client, such as bills and car-rental preferences. Would you apply the same design decision in this case as in the previous question, i.e. how would you ensure data modification on client profiles? Provide a short motivation.

We could use a Client class that keeps track of client profiles. This would get updated with every transaction.

3. Consider the following two alternatives to implement transactional semantics of the first question: (i) relying on transactions only (i.e. this requires XGT) (ii) using application logic to rollback if the quote confirmation (partially) fails. Discuss their difference in locking (concurrency control): How does each one prevent concurrent changes to a particular set of data?

When relying on transactions only, we rely on the google Datastore wich uses optimistic concurrency control. It works by checking the last update time for the entity group in the transaction. When that update time is changed on committing, the transaciton will fail because it means another transaction has modified the entity group in the meantime. When using application logic we have to use our own concurrency control.

For large-scale operations, optimistic concurrency control would be optimal for high responsiveness.