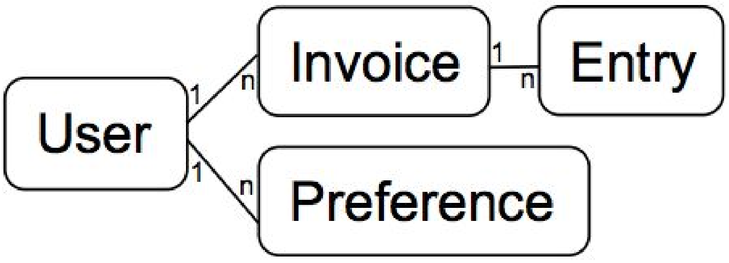
Ilias Bachiri r0747118
Jasper Maes r0763593

Google App Engine part 1: Design discussion

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.

In our design, the CarRentalCompany is the root entity, which make that there 32 entity groups. Cross-group transactions can perform a query on a maximum of 25 separate entity groups. We can Cross-group transactions however it adds some latency to the transaction and since we only have 32 entities, we prefered the normal transactions.

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.



In this independent part, the Client can become the root entity. Realistically there will be many more clients than company's, which would make XGT's more beneficial, so in this part we would make a different design decision.

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?

- **Cloud applications typically aim at large-scale operation, facing many concurrently accessing requests. Yet, as a crucial success factor, high responsiveness must be ensured at all times. Which alternative of those you discussed above would you prefer?**

(i) If we rely on entity groups, this makes the possibility of an update way less frequent. In this case we also have optimistic concurrency control. (ii) With application logic there is no-locking, we have optimistic exception driven concurrency. When one transaction fails, the exception secures that all the transactions are rolled back.

- We would prefer rollback by application logic since XGT is only really viable for a small amount of entity groups concurrently. We would sacrifice responsiveness otherwise. Although the trade-off will be a more complex application logic.