**Post Production Notes**

We encountered a few problems during the design of the database system which we did not predict, either through a lack of knowledge or insufficient analyzation of the problem. For example, nowhere in the initial specification did we describe how the application would eventually communicate with the database in terms of getting individual entries in the relation. Even though we came up with what we find to be an extremely convenient solution to this problem, it could have been specified earlier. Throughout these notes, we will describe some further difficulties, and present how we learned from them through this process.

At the beginning of this project, we had never gone through the sort of product design cycle as this project called for before. The result is that even though we came up with what was graded as an excellent design document, we can look back and point out serious failures on our part which could have been fixed. Something in particular I look forward to getting right on the next project is identifying the problems better. That requires a more in depth look at all parts of the final product before diving into the specifications. A prime example is the way no one in our group was able to identify storing conditions as a serious problem. In the end, that simple (or not so simple) aspect of the project required two classes of its own, and even though we created what can be described as an elegant solution to that problem, solving that problem ate up some of the time we had to create the database engine. The result was that the first week, during the design document, we did significantly less work than we did the following week, where we crammed up to the night it was due to put some polishing touches on the engine. As stated earlier, this is something that was very educational, and even though I would describe our problem as excusable for the first project, it will be interesting to see how effective we are in limiting that problem for the next projects. The main point can basically be boiled down to the fact that each week of the project should demand an equal workload.

Another example of the lack of specification we had was in designing the parser. We could have identified much more specific functions for the parser, and had we taken the approach of creating the basic parsing functions (parsing an attribute list, identifier, condition list, etc.), we could have done a much better job of coding the parser from the bottom up. In addition, in the initial documentation we did not specify a good way of returning invalid input. This ends up being a significant design issue, as we found when were almost done with the parser. What we ended up with was a system where many of the parser functions returned and integer to signify INVALID or SUCCESS, which was a solution we came up with about halfway through the coding process. That meant that afterwards, we needed to go back and standardize all the code to our new specifications. Once again, it meant spending time in solving the problem, and in this case fixing old code, while we could have been working on the next deliverable.

A final example of insufficient specification came with the design of the application. A specific requirement was to use all of the relational algebra functions in the application, so ideally in the design document, we would have at least mentioned every operation somewhere in the application specification. This was definitely not done, and in the end it meant adding to the application design in order to have relations which allowed us to naturally perform those operations.

**Individual Work Load Distribution**

The work load for this project was spread out evenly in general. Each member of the team had a specific role for each part of the assignment. For example, for the engine and parser, Cameron created the structure and design of the classes, and laid out the functions that needed to be done. Chris on the other hand started learning how the unit tests worked and the specifications they needed to meet. Taylor provided general coding help to both other members to make sure each part was getting done. Then for the application, Taylor and Chris took the lead in the organization and specification, while Cameron did polishing work on the old files. Chris also went through and created more tests, while Cameron was then able to go through and view the application with unbiased eyes (as he had little to do with its creation). This allowed him to provide a good description and make effective comments to the usability factor of the application.

If the distribution must be put down into a percentage, lines of code or gitHub commits is definitely not an effective way of describing work done (take, for example, fixing 10 bugs, which taken a couple hours but only gives 20 lines of code on gitHub). Everyone in the group is comfortable describing the work as approximately a three-way split.

Cameron – 33%

Taylor – 33%

Chris 33%