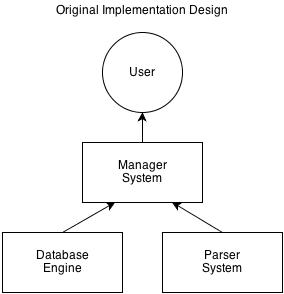
Post Production Notes

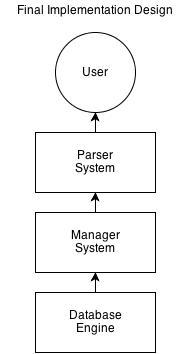
Report

Design Changes

Key design changes we made within the structure of our database that strayed away from our original design included the following:

1. Our integration of the parser system and database engine did not go as we had planned. In this sense, we had our validation function call the manager functions which actually applied the intended command to our database. In our previous design, we had anticipated that our manager class to essentially be the interface between the database engine and the parser system. This turned out to not be the case when we determined that our parser did not integrate nicely in such a fashion. Our solution to this problem was to simply rework how we planned to integrate the parser system and database engine. In the new design, we had the parser envelop the manager system; in essence, it still acted as an intermediary between the database engine and the parser system, however the point of access to and from the user was changed. In the future, we plan to design our code around the overall goal rather than focus primarily on having each individual piece as a separate entity. These two different integration methods are summarized in the figured below.





1. A miscommunication issue that we ran into involved how the parser system was supposed to be implemented, more specifically, our validation function. Conflicting instructions given led to a dichotomy in how commands were lexicographically analysed. One methodology, taking the hands-off approach, involved simply reading in the command, checking for validity, then returning a result; the other methodology, which took a more direct and hands-on approach, consisted of checking for validity and then removing irrelevant substituents from the string. For example, once the string command “INSERT INTO animals VALUES FROM ("Joe", "bird", 2);” was parsed into the section of code which handled insertion, the substrings “INSERT”, “INTO”, “VALUES”, “FROM”, etc. were removed and the remaining string “tidied up” before the relevant information was passed onto the database engine through the manager system. In the end, post-integration, the result was that all the work had to have been done on the string regardless and where we handled it, whether in the parser system or the managerial system, was irrelevant.
2. See included design document for updates

Lessons Learned

1. Communication is essential to a team project. This does not just mean clarity in what tasks each member is responsible for, but also that each team member has an understanding of what is happening to the code base. The latter part is important because unless each member has active involvement in the code base, they won’t be able to contribute as quickly or effectively.
2. When designing the overall structure of the code, it is better to actually create the skeleton code rather while writing up the design document. If a code structure is built first to encompass all functionality, design flaws can be caught earlier and then accounted for.
3. While programming the actual code, it is necessary to keep in mind how each class, each code and every function will interact with the other components of the program. This will ensure that code remains neat and is able to realize the desired functionality.