# Post Production Notes

There were many changes that we had to make to our design so that we could progress with the requirements of forthcoming deliverables.

* **Change 1:** *Changed how primary keys work*
  + Previously, primary keys were indicated in a separate vector which indicated the columns that were primary keys. However, keeping a separate vector was inefficient and determining whether a column was a primary key was difficult because another vector had to be searched. The change was to integrate the primary key as a Boolean into the column entry’s tuple, along with the existing key and type. By doing so, determining the primary key status of a column became significantly easier and faster.
* **Change 2:** *Changed how the parser worked*
  + Previously, our parser saved tables by writing the display output into a file. However, this did not meet the project’s requirements and parsing the display output was difficult. Our change was to keep a stack of the inputted commands of the present session, and pop the lines of commands into a file upon writing. This allowed us to reuse existing parsing functions to read the file instead of writing new ones to read a displayed output.
* **Change 3:** *Changed the functionality of most Engine and Table functions*
  + Many of our functions within Engine and Table formerly operated without regard to table name output, operated without regard to Parser requirements, or did not allow for complex situations. We had to modify the functionality of these functions so that they would work with parser. Some examples include rename and project, where previously they self-generated an output table name, or selection and deletion, which only accepted a very simple condition, or union and difference, which did not accept relations with the same column content if they were in different orders.

There were several difficulties that we encountered during the development of this project that slowed our progress.

* **Difficulty 1:** *Keeping track of relational information*
  + Initially, we were unsure of how to store relational information or the data structure to use, because we wanted something to hold our content’s name, be versatile with the data type (VARCHAR, int, etc), and be able to easily link with the key and the primary key flag. Eventually, we settled upon tuples for our content information (key (int), name (string), type (string), primary key flag (bool)), because it allowed us to easily store any kind of information, be versatile with different types supplied by the user, and easily join the information together.
* **Difficulty 2:** *Creating new tables from saved files*
  + When we originally saved new tables, we had an issue of reading the output file back into the engine. Our previous method of writing to a file was to direct display output into a file, but that made parsing difficult. Eventually we modified it so that it output parser commands, which allowed us to use existing parsing facilities to read the file.
* **Difficulty 3:** *Addressing parser issues*
  + There were several difficulties that we encountered while developing our parser. We
* **Difficulty 4:** *Splitting up the work*
  + Splitting up the work was not easy, mostly in part due to our poor planning. Sometimes the work was arbitrarily assigned or self-volunteered, which resulted in some people doing more work than others. The solution to this is planning ahead and getting solid structure for our database in documentation.
* **Difficulty 5:** *Conditionals*
  + We originally developed our functions to use very simple conditions such as “a > b” or “c = d” but we struggled with developing a solution to complex conditionals, such as “a > b || c > (d \* e)”. The solution to this is developing a tree structure of conditionals, which allow the individual elements to be resolved within complex statements.

Throughout the development process of this project, the changes that we’ve made and the difficulties that we’ve encountered have taught us many lessons.

* **Lesson 1:** *We need to start early and plan ahead.* 
  + When we first developed our original design document, we had no plan this showed through our low score. Our lack of planning in the early stages resulted in unclear requirements and expectations of our program and incomplete functionality at the time of submission. Without a clearly designed document and API, we wasted lots of potential development time trying to create a structure for our database program.
* **Lesson 2:** *We need to develop with a unit test framework and IDE*
  + Although we were able to develop code easily without an IDE, the requirement for a unit test framework through an IDE like Visual Studio made our initial integration difficult and time consuming because we did not understand the framework’s structure or functionality. This resulted in submissions without proper unit tests.
* **Lesson 3:** *We need to utilize debugging tools*
  + When we first integrated the unit testing, we were snagged with an issue in our parser that did not allow it to work. When we did not use the proper debugging tools, we were unable to determine the cause of test failures. Only until we began using the debugging tools within Visual Studio were we able to correct the issue.