## Requirements

The requirements will outline what the application *must* do at base level, both from a user perspective and an operational perspective. This document will be outlining the requirements of the baseline project, though the scope of the project can and may be expanded at a later date. The scope increase generally should not interfere much with the baseline requirements presented by this document.

There are several aspects about how the application will operate that has not yet been decided. The UMLS and requirements are a loose draft of the expected flow of the application, though this may change drastically during development as more efficient ways to approach the project are understood.  
  
Please view “Requirements Appendix” folder for the UMLS associated with this document.

### Prerequisite - The Challenge of Indirectly Modifying a Database

There is a particular challenge within this project that may change drastically how the UMLs lay out the applications inner workings once resolved. This problem arises on the fact that this application will is intended to allow a user to work on a database *indirectly*. This meaning that if a user were to modify a database within the application, the modifications are not applied unless the user directly decided to save. This brings up a particular challenge in how the application is implemented, and there are two routes I have conceived to navigate this issue, but neither make the process easy.

In this requirements document, the UMLs are such that we assume new queries are sent to the database when a user loads, reloads, sorts or view selects a table with any modification to the table being saved within the application. When a new query is sent, such as to order, view or filter a table, the new modifications would have to be accounted for and positioned in the query appropriately. This also brings up a bigger challenge when a user has changed the primary keys or if no primary keys are present, as changes to entries cannot be so easily mapped to the unmodified database.

An alternate approach would be to load an entire table once and have it be sent to a lower layer when additional ‘queries’ are used (sort, select, etc) and the data loaded directly from this layer. The issue with this approach is it creates complications such as a user not wanting to load a whole table thus making it take up more time than necessary. Additionally, things such as sorting data or filtering data would be handled within the application and not the database, which may make it take longer, depending on how it is handled.  
  
This will not be attempted to resolve within this document, but it is a critical challenge appropriate to the assumptions on the implementation presented here that will be addressed at a later date.

### Functional & Non-Functional Requirements

#### Functional

* System allows user to enter server credentials and create a connection if the credentials prove valid.
* The system must allow for databases to be opened, created, modified, and deleted a database should any security credentials be met.
* User must be able to edit a database without consolidating changes unless explicitly requesting to save.
* If a user has the requirements to modify a database, they cannot open another database while unsaved changes to the current database have been saved or removed.
* User can view, sort, view joins and filter table views. Entries can’t be edited here but can be selected and edited separately.
* User can add multiple parameters to their sorts, viewing and joins from drop-down menus.
  + Options from the drop-down menus such as selecting specific fields are created from the specific database the user is in.
* The system will ask a user to confirm when saving or exiting the application (with unsaved changes).

#### Non-functional

Prerequisite: computer the application is run on must be able to establish a connection to an SQL server (local or remote) to use the application.

* The application will attempt to create a connection to the SQL server specified from which it can then pass queries to, and handle data received.
* Database access levels are specified within the server itself, so the application will follow accordingly.
* Since it is a “middleman” program, it must be able to handle errors that occur when inputting/outputting data into/from the database and present necessary information to user.
* New database data created from the application by the user must be able to be passed into the database.
* When user requests to save, the application must check for NULL fields or duplicate IDs and reject a save attempt if the database.
* Modified and new (and ONLY modified/new) entries are stored temporarily within the application (non-modified ones are not stored, only fetched).
* When requesting to save, the application will reject a save attempt if:
  + Duplicate keys are identified in a table (application data).
    - Including constraint keys (check database).
  + NULL entries when database rejects them (check database).