Phase D – Physical Design

Table Creation Scripts

The create table scripts are modelled directly after the ER diagram present in the previous section (or to be more precise, the model was made after the tables in this section). Therefore we will not go over the code for every single table, as it is simply the sql code for implementing the ER diagram shown before. There are some special mentions, however:

gre

```
check (verbal>=130 and verbal<=170
and quant>=130 and quant<=170
and analytic>=0 and analytic<=6),
```

As the scores of gre must be set in a specific range, I made sure to use some sql check statements to prevent the user from inputting incorrect scores. However, as I learned much too late, the MySQL we are using does not support check statements, so it doesn't actually do anything. But it's the thought that counts, I suppose?

education/application

I did consider making education's degree attribute and application's semester an enumeration, between (bs,ba,ms,ma) and (spring,fall), respectively. However, as the user interface will offer a multiple choice anyway, I decided not to double the limitation in SQL. This also helped during implementation, if I decided to change anything; I wouldn't have had to do it in multiple places.

education.gpa

The GPA value is limited to the normal range of 0-4. I considered not having this at first, as there might be a time where a different GPA scale is being used. However, since we do compare the GPAs in a later query, decided that GPAs must be forced to be on the same scale. But again, apparently, MySQL does not support check statements, so it doesn't actually do anything.

degree_name primary keys

As recommended by the requirements, the degree name was made the primary key. This might have been a bad idea, as the degree name is not only used as a foreign key in many other tables, but as a usable data field in degree object. As such, any changes to this degree name must be cascade through the database. Looking back, I probably would have changed degree_name to not be a primary key, and instead used a unique constraint on it. I realised this a little late, and don't really want to refactor the entire database.

Table Deletion

Not much special going on here, except that I tried to make sure tables are dropped in reverse order from when they are created, so that foreign keys don't fail.

Data loading Scripts

The data loading scripts were created by using the program to enter in data, and then exporting the data with MySQL Workbench's data export feature, which could export the data as a file of sql statements. I decided to do this instead of making up insert statements myself as the database program handles the incrementation of IDs for me, as it was designed to.

User Interface Design

As we are allowed to make a pretty bad user interface, which will be thoroughly described later, I decided to go with a text based interface. By navigating through menus and submenus the user should be able to access and edit information for all the required tables without having to memorise too many things. I did not want the user to have every choice up front, that would make it too confusing, and I didn't want to make a command line interface, that would be too much memorising.

Denormalisations

Denormalisations for this project seemed most relevant to the required queries we are supposed to do. However, for general usage - add, update, view, delete, and essentially everything do-able by navigating the menus of the program, the normalisations in place are effective. I tried to make sure that to get relevant data for most menu screens, the SQL connector does not have to select from multiple tables. Of course, for some of these transactions, notably the requirement-answer and criteria-evaluation transactions, in order to display the list of questions and answers alongside each other, the query had to access both tables. However, I decided to keep these tables normalised, as if you count the number of operations performable on each of the tables individually (editing, adding, ect.), it obviously outnumbers that of the operations performed when they are together (only displaying information alongside each other). For the queries, many of the queries will need access to multiple tables to accomplish their task. As we don't really have that much data, so the queries will be fast, and I ultimately decided that the update and maintenance of data is more important than the queries, I decided not to denormalise anything even for the queries. That, and also the reason of I didn't want to mess up my tables any more.

Secondary Indexing

I decided not to create any additional indices, as the database at its current state is not expected to be very complex, and as it is rather normalised in my opinion. Just by having primary keys the database should operate at an acceptable speed.

User Requirements Satisfaction

As these tables were designed to fit the ER diagram shown in the previous phase, which was created a transformation of the ER diagram in the previous-previous phase, I would think that the data requirements are still being satisfied by this design. I even kept education and GRE scores attached to application, although it might not be a good idea, to ensure that that particular requirement is not being skirted.