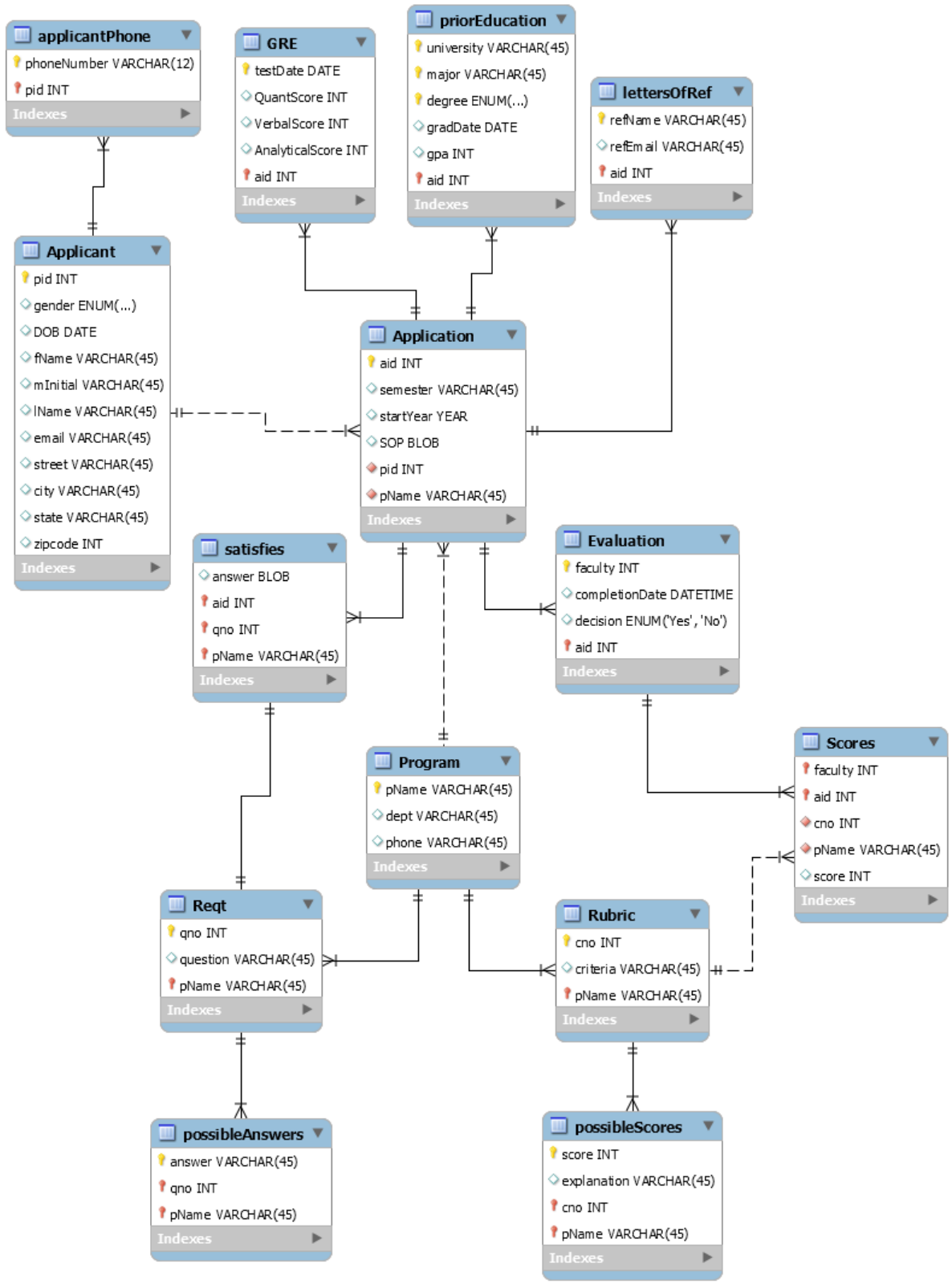
CMSC 461: Final Project Documentation

Jupyter Database

A graduate application based portal

Brianna Richardson

Relation Diagram



Logical Design:

Within the above table:

- Application is in a many-to-one relationship with Application to provide Applicant the option of multiple applications.
- GRE, priorEducation, applicantPhone, lettersofRef became their own table with a many-to-one relationship with either Applicant or Application because they were multivalued.
- Program links to application in a one-to-many relationship because there will be multiple applications for one program
- Program links to Rubric & Reqt with a one-to-many relationship because each program can have more than one additional requirement and rubric criteria.
- Each rubric & Reqt has possible responses depicted by their own table to show their multivalued state.
- An additional requirement is satisfied with an application by the Satisfies table, which includes the response to the additional requirement by a certain application.
- There are several evaluations from different faculty that contain scores for each rubric criteria. Therefore, score is its own table, as well.

Attributes of each table, with stars beside key attributes. Please note attributes of the same name are foreign keys:

- ApplicantPhone: phoneNumber*, pid*
    - Applicants can have multiple phones, so each phone is unique based on the number and the owner
- GRE: testDate*, Qscore, VScore, AScore, aid*
    - Applicants can take the GRE multiple times, so each GRE is unique based on when it was taken and by whom. Cannot take the GRE twice on the same day, so each date for each person will be unique.
- priorEducation : university*, major*, degree*, gradDate, gpa, aid*
    - Applicants can have many previous degrees. These degrees can be in the same major as another degree and can be from the same school as previous degrees. For example, an individual can have a BS and a MS in Biochemistry from Towson University. The degree is the difference maker in this situation. But since you can't get two of the same degrees of the same major from the same school, this selection of primary keys should remain unique.
- lettersOfRef: referenceName*, referenceEmail, aid*
    - Since reference writers can write multiple references and applications need more than one reference, it makes sense to make both my primary key. Since one reference cannot write more than one letter for an applicant, this is a safe set of primary keys.
- Applicant: pid*, gender, DOB, firstName, middleinitial, lastName, email, street, city, state, zipcode
    - Each applicant will get a unique person identification
- Application: aid*, pid, semester, startYear, SOP, pName

    - Each application will have it's own identification number
- Satisfies: answer, aid*, qno*, pName
    - Each additional requirement will permit only one response by an application.
- Evaluation: faculty*, completionDate, decision, aid*
    - There are several evaluations on a given application by different faculty members. Therefore, each evaluation will be unique on who evaluates what, ie faculty and application.
- Program: pName*, dept, phone
    - Each program has a unique program Name
- Scores: faculty*, aid*, pName*, score
    - Each rubric criteria will have a score given by a specific evaluator. Therefore, for each critera from a specific program, there is an faculty who provides a score.
- Reqt: qno*, question, pName*
    - Each requirement is given a requirement number and it is from a specific program. Therefore, these are the identifiers.
- Rubric: cno*, criteria, pName*
    - Each rubric criteria is given a criteria number and a program name to separate it from other crtierias within the same program and across programs
- possibleAnswers: answer*, qno*, pName*
    - There are several possible answers to different requirements from a given program, so these must be created keeping each of those aspects in mind.
- possibleScores: score*, explanation, cno*, pName*
    - There are several possible scores to different criterias from a given program, so all primary keys are needed to maintain unique-ness.

Normalization:

1NF: Each table is in 1NF because:

- There are no duplicates because primary key restrictions
- Everything in an attribute column are of the same type due to type casting restrictions.
- Each cell is directly inserted into the database, so there are no columns with the exact same input (unless NULL)

2NF: Each table is in 2NF because:

- They are in 1NF
- Nonkey attributes can only be determined by all key attributes:
    - Proof by example for the Evaluation table
    - Primary keys are: faculty & aid
    - Non-primary key: completionTime, decision
    - Cannot get tuple without all primary keys:
        - Faculty alone: will give all evaluations done by a faculty
        - Aid will give you all evaluations done for an application
    - Therefore, all primary keys required

3NF: Each table is in 3NF because:

- They are in 2NF
- There are no columns that are not dependent on primary keys

Physical Design

The physical Design is implemented in the attached code. Each of these codes run with no errors:

- createAll.sql: creates the tables. This can be run in mySql and it can also be implemented in Python!
- loadAll.sql: loads in a set of data to work with. The following tables are loaded as examples:
  - Application
  - Applicant
  - GRE
  - priorEducation
  - lettersOfRef
  - Evaluation
  - Program
  - Rubric
  - Reqt
  - possibleAnswers
  - possibleScores
- queryAll.sql: if the above functions are run, this function will query out information based on the functional requirements provided in the project documentation.
  - With the exception to the 7^{th} request, all these requests work as expected.
- dropAll.sql: will drop all tuples and all tables from the database.

Prototype, Development, Testing

The development and testing of the database is done through the Python code: **runProgram.py**. This program runs the queries from queryAll, within the Python space. It asks for user input and creates the output based on the requests of the user. It is a clean way to implement the queryAll. Within this code, execute statements can be written to insert, update, and delete from the database.

User's Guide

Thanks for servicing '***Brianna's 461 Database Project'***! Within this application, a database has been created so that YOUR school, Jupyter College, can better navigate the application process. Our services have provided both the Sql database and some example insertions and queries that will allow you to build into the Jupyter database based on the functional requirements that you had. Here is a short tutorial for how to utilize this tool.

Instructions:

1. Follow the instructions here (https://www.csee.umbc.edu/~kalpakis/courses/461-fa17/project/MySqlGuide-461.pdf) for how to download MySql
2. If you're interested in incorportating the Python feature, follow the instructions here (https://medium.com/@GalarnykMichael/setting-up-pycharm-with-anaconda-plus-installing-packages-windows-mac-db2b158bd8c) for how to download PyCharm.
3. In your MySQL workbench, load the provided files: queryAll.sql, loadAll.sql, createAll.sql, and dropAll.sql
4. Run the createAll.sql code to establish the database and the tables.
5. Following the set up of loadAll.sql, users can input their own tuples.
    a. If you want to replace the tuples loaded in the loadAll.sql, try commenting it out or create your own! With each file you create, don't forget to include the use `jupyter` at the top!
6. If you want to remove a tuple, check out the dropAll.sql file. Keep in mind that dropAll.sql will remove all tables so if you want to load tuples, you will have to create the tables again, as well.

Again, thanks for servicing, '***Brianna's 461 Database Project'***!