Khang Ngo
CMSC 461

CMSC 461 Project Phase A & B Report

The project is an implementation of a SQL database powered by Python and intractable by a user to perform a specified set of actions, while following a series of steps in order to ensure maximum familiarisation with the processes involved in the designing of a database.

Project Phases Analysis

Phase A of the project consists of an analysis of each phase of the project as well as analysis of the requirements of the database server to be created. By analysing the phases of the project in depth, one can learn more about and better understand the various steps of designing a database server.

Phase B of the project involves the creation of ER diagrams identifying relationships, entities, attributes and constraints that become the design of the final database's tables. This phase involves close analysis of the project requirements in order to create entities containing the correct required attributes and relationships.

Phase C of the project is the mapping of Phase B's ER model to actual tables that will be used in the database server. Normalisation skills will be applied in this phase to produces the most efficient tables possible whilst retaining satisfaction of project requirements.

Phase D is the writing of SQL scripts to create physical SQL tables based on designs made in earlier phases. This phase will test SQL writing skills and understanding of the ER model and tables designed in previous phases. The requirements must be well understood in order to write efficient scripts retrieving the correct required information from the tables.

Phase E is the development of the Python interface which will interact with the SQL database and perform required actions, as well as indexing of the database. This phase will require skills on how to properly use the SQL database now that it has been created.

Requirements Analysis

The database to be implemented is an academics-assisting database designed to process graduate applications to a college. Applications contain a variety of information and are ultimately evaluated by a professor and given a final decision. The requirements specify a list of queries the database should be able to make, and most of these queries seem to be statistical data retrievals of the applications in the database.

Data Requirements Analysis
What required attributes I determined each required entity should have, based on the project descriptions.

Applicant
- student ID
- contact information
- birthday
- name
- gender

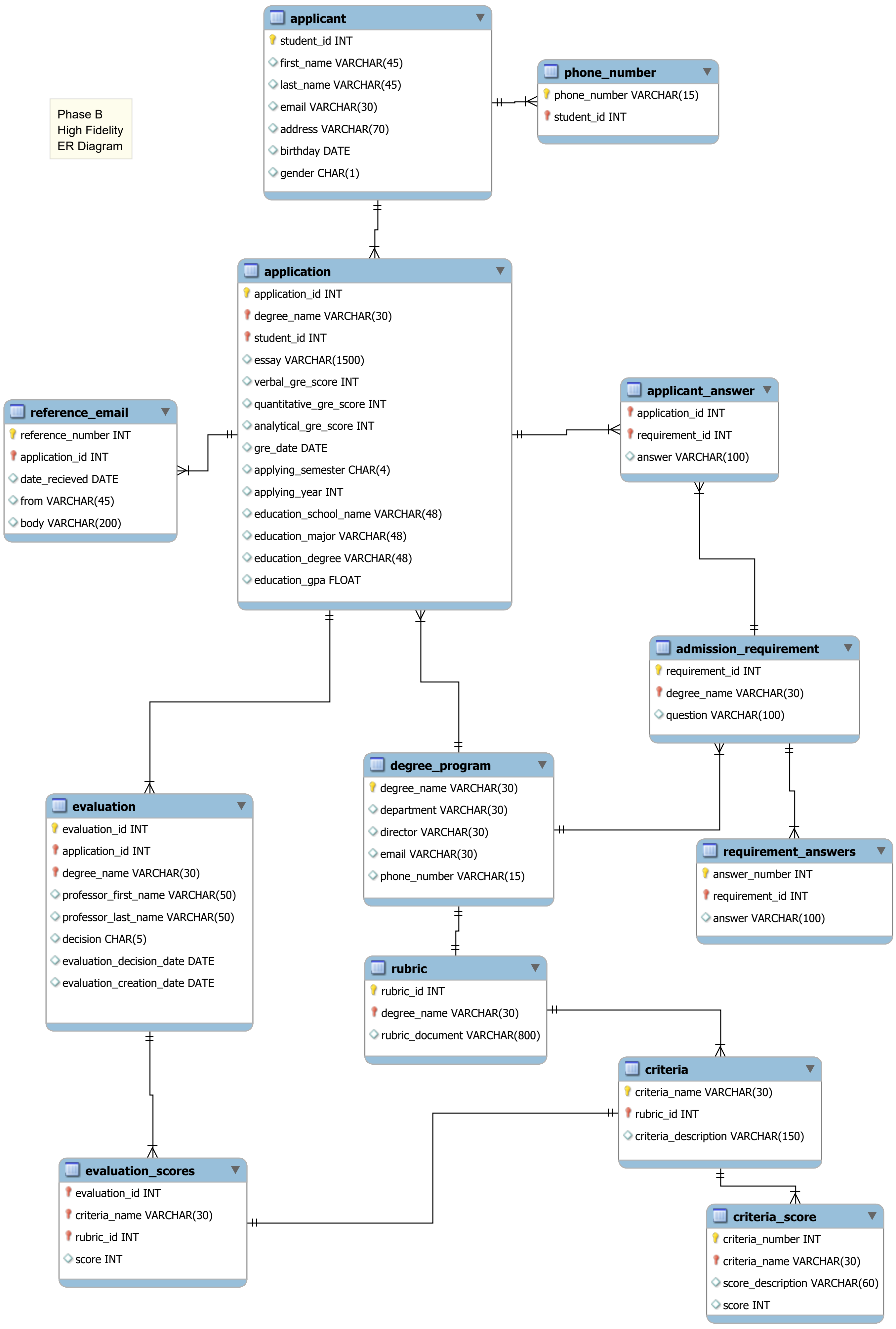
Application
- degree program
- GRE information
- essay
- requirements answers
- semester
- year
- reference emails
- education

Degree Program
- name
- director
- email
- department
- phone

Admission Requirement
- question
- possible answers

Admission Rubric
- evaluation criteria
- criteria possible score and description

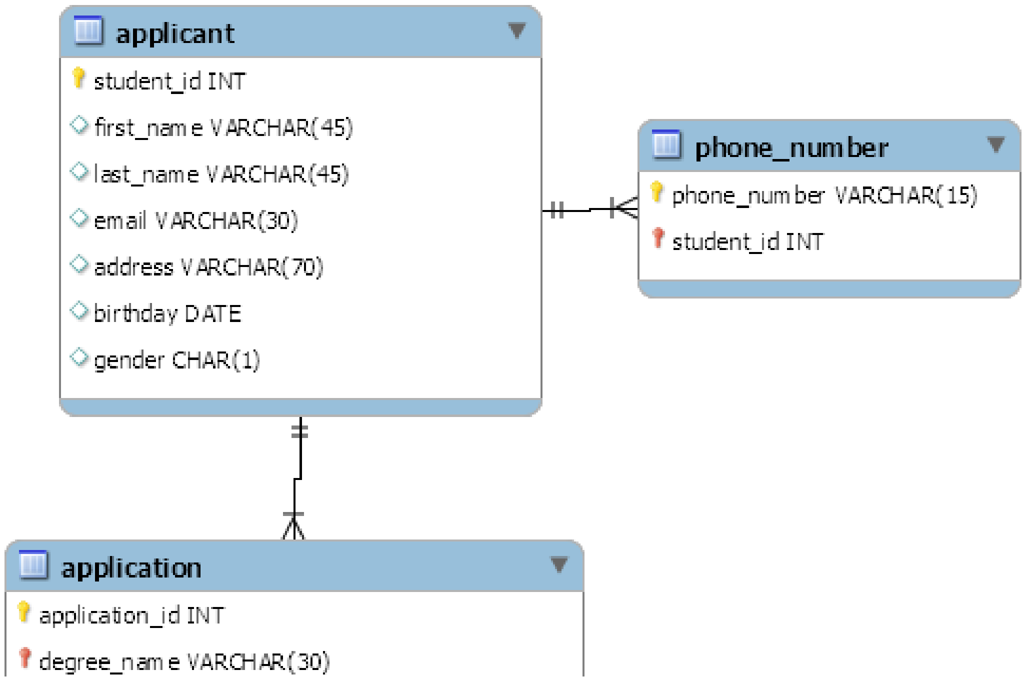
Admission Evaluation
- professor [making evaluation]
- decision
- degree program
- criteria scores
- date/time for evaluation decision and evaluation creation


Phase B
High Fidelity
ER Diagram
applicant
student_id INT
first_name VARCHAR(45)
last_name VARCHAR(45)
email VARCHAR(30)
address VARCHAR(70)
birthday DATE
gender CHAR(1)
phone_number
phone_number VARCHAR(15)
student_id INT
application
application_id INT
degree_name VARCHAR(30)
student_id INT
essay VARCHAR(1500)
verbal_gre_score INT
quantitative_gre_score INT
analytical_gre_score INT
gre_date DATE
applying_semester CHAR(4)
applying_year INT
education_school_name VARCHAR(48)
education_major VARCHAR(48)
education_degree VARCHAR(48)
education_gpa FLOAT
reference_email
reference_number INT
application_id INT
date_recieved DATE
from VARCHAR(45)
body VARCHAR(200)
applicant_answer
application_id INT
requirement_id INT
answer VARCHAR(100)
admission_requirement
requirement_id INT
degree_name VARCHAR(30)
question VARCHAR(100)
degree_program
degree_name VARCHAR(30)
department VARCHAR(30)
director VARCHAR(30)
email VARCHAR(30)
phone_number VARCHAR(15)
requirement_answers
answer_number INT
requirement_id INT
answer VARCHAR(100)
evaluation
evaluation_id INT
application_id INT
degree_name VARCHAR(30)
professor_first_name VARCHAR(50)
professor_last_name VARCHAR(50)
decision CHAR(5)
evaluation_decision_date DATE
evaluation_creation_date DATE
rubric
rubric_id INT
degree_name VARCHAR(30)
rubric_document VARCHAR(800)
criteria
criteria_name VARCHAR(30)
rubric_id INT
criteria_description VARCHAR(150)
evaluation_scores
evaluation_id INT
criteria_name VARCHAR(30)
rubric_id INT
score INT
criteria_score
criteria_number INT
criteria_name VARCHAR(30)
score_description VARCHAR(60)
score INT

Design Decisions and Assumptions

Diagrams shown below are not a substitute for the entire diagram. They are intended to focus attention on a particular part of the entire diagram, please continue to refer to the whole diagram when reading descriptions.

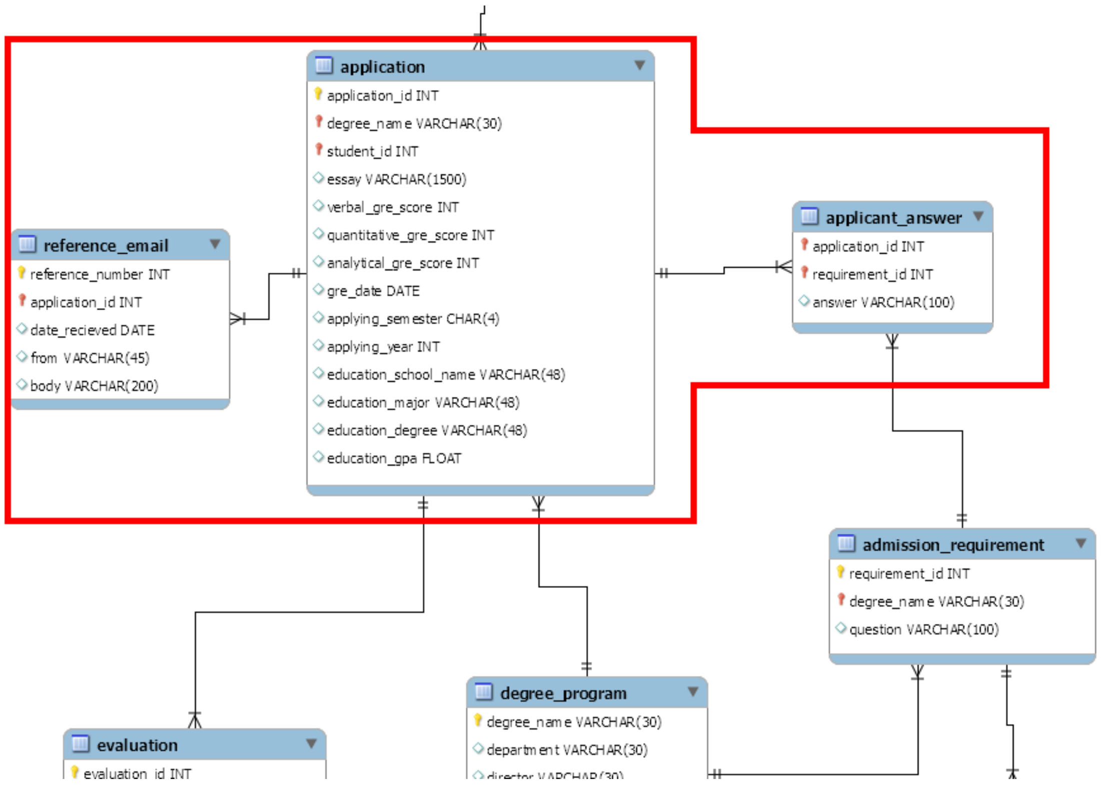
Applicant



Assumptions: An applicant can make multiple applications, as such, the applicant entity is in a 1-to-many relationship with the applicant entity. Applicants can only have 1 address and email.

For the applicant entity I decided to split the name into two attributes, first name and last name, as it may be convenient to search for a student without needing the full name, in some cases. Phone number must be stored in its own entity as it should be possible for an applicant to have multiple phone numbers. Although the image above shows the applicant being in a 1-to-many relation with phone_number, alternatively I could have made it a many-to-many relationship, allowing for some applicants to share phone numbers. Due to the composite primary key I have decided for phone_number, this conversion should not be problematic should the need arise. The primary key for phone_number is the combination of the actual phone_number and the foreign key student_id to which the phone number is attached. This way sharing phone numbers will still produce unique entities, as the student_id foreign key will be different for each shared number.

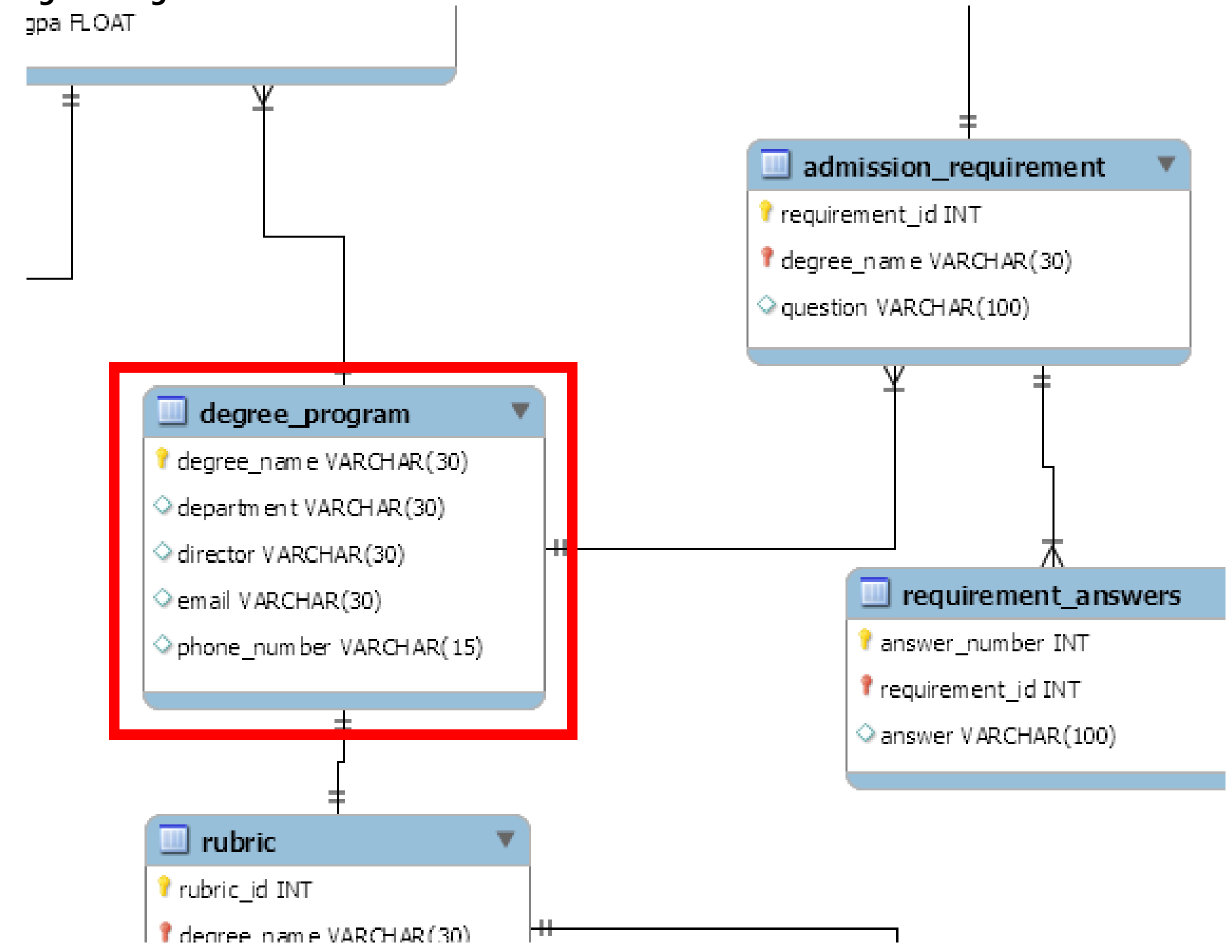
Application



Assumptions: Applications can only have 1 essay. Applications can have multiple educations and multiple GRE scores attached, although it is not represented in the high fidelity ER diagram.

I determined that the application entity would be one of the core entities of the database system, and as such it is related to many other entities and contains a high number of attributes. The primary key of the application entity will simply be its application_id, a unique number produced for every application created. Reference Emails and answers of the applicant were placed into separate entities as it was possible for many of them to exist. The primary key for an applicant_answer is the combination of the foreign key application_id of the application it belongs to, and the requirement_id foreign key, which specifies which requirement, and by extension which question, the answer entity is answering. Since each applicant can only have one answer to each requirement question, this should allow all answer entities to remain unique. The primary key for a reference email will be the combination of the application it belongs to and a reference number, which might be relative to the application instead of globally. This allows reference emails to remain unique from each other, even if they have the same sender or were received on the same date. In the diagram shown above, GRE information and education information has been placed within the application entity. However, it may also be possible for an applicant to submit multiple education entries and/or multiple GRE scores, in which case they could be separated into their own entities. However, as the requirements did not explicitly state that an application should be able to hold multiple education/GRE information instances, I kept them together for now.

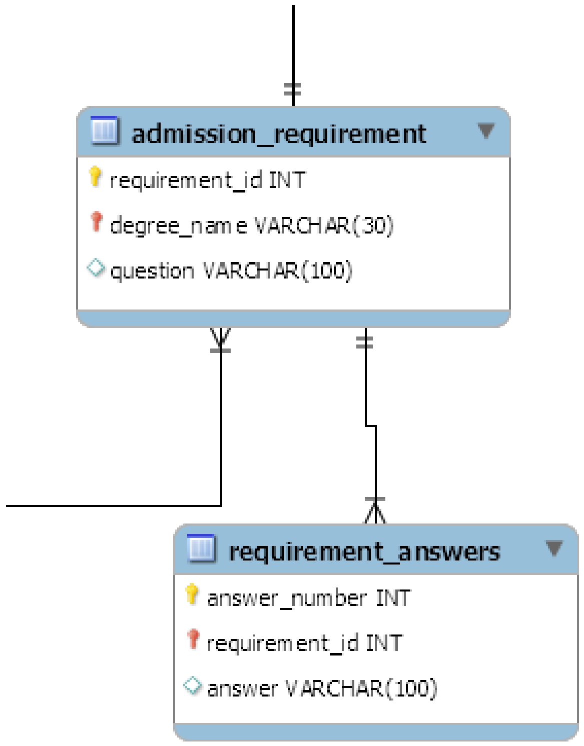
Degree Program



Assumptions: A degree program can only be part of one department, have one director, email, and phone number.

A degree program is identified by its primary key degree_name, which is just the name of the program. The requirements state that the name will be unique, which is what should allow this to be possible.

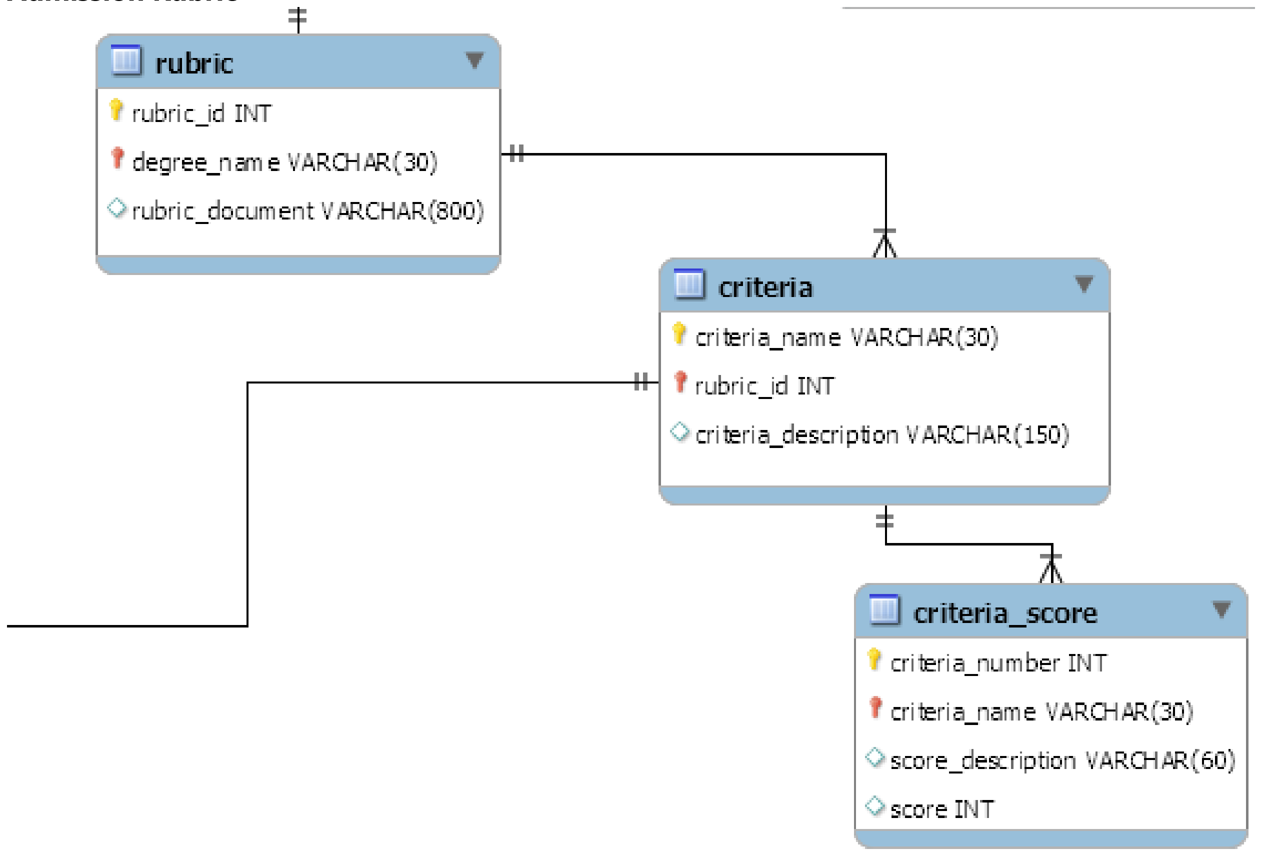
Admission Requirement



Assumptions: A requirement entirely is composed of just the question. Although I personally don't actually think this makes sense, the requirements seem to imply a question is a requirement.

The requirement entity is identified by its requirement_id, which I will have be unique to the whole database. Alternatively, I could make the primary key a composition of degree_name and requirement_id, and have requirement_id be relative for each degree_program. If it becomes clear that implementing a requirement entity with a relative ID is simpler, I may opt for that instead during implementation time. The requirement entity may contain multiple answer entities, identified by the composite key requirement_id and answer_number. As requirement_id will be unique, answer_id will be able to be relative to each requirement.

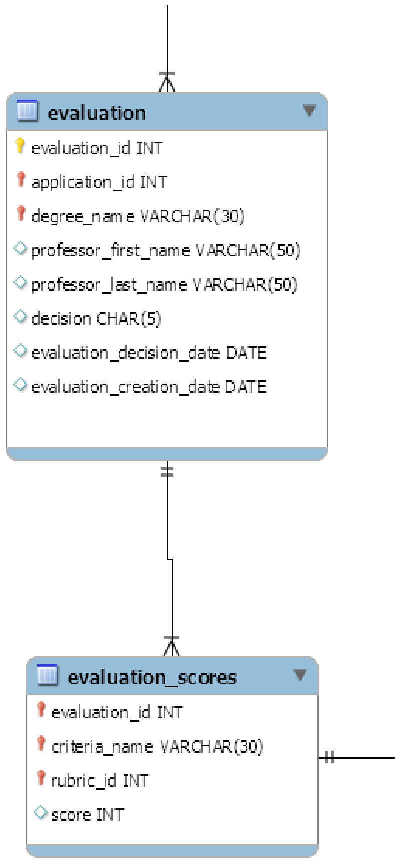
Admission Rubric


rubric
rubric_id INT
degree_name VARCHAR(30)
rubric_document VARCHAR(800)
criteria
criteria_name VARCHAR(30)
rubric_id INT
criteria_description VARCHAR(150)
criteria_score
criteria_number INT
criteria_name VARCHAR(30)
score_description VARCHAR(60)
score INT

Assumptions: Each rubric can only belong to 1 degree program.

I modelled the rubric system after the rubric described in class by the professor. His rubric contained a set of criteria, each criteria of which had certain descriptions that described a score. As such, each rubric is able to have multiple criteria entities, identified with a unique criteria name and the rubric_id it is related to. Criteria can also have their own description. Each criteria then can have its own set of criteria scores, which contain descriptions of what a student must to do to fulfil that particular criteria score. Criteria scores are identified by their primary key combination criteria_number, which is actually more like a counter for each possible score for a criteria, and the foreign key criteria_name, which specifies which criteria the score is describing. Additionally, the rubric may contain a document summarising the rubric and requirements overall; I find rubrics are often accompanied with such in real life.

Application Evaluation


evaluation
evaluation_id INT
application_id INT
degree_name VARCHAR(30)
professor_first_name VARCHAR(50)
professor_last_name VARCHAR(50)
decision CHAR(5)
evaluation_decision_date DATE
evaluation_creation_date DATE
evaluation_scores
evaluation_id INT
criteria_name VARCHAR(30)
rubric_id INT
score INT

Assumptions: There can only be 1 professor for each evaluation, although multiple professors can submit separate evaluations for a single application.

The application evaluation entity's primary key consists of a unique evaluation id and the application id of which it is related to. Evaluation entities can have multiple evaluation score entities, which represent the given scores for criteria in the rubric of the degree program the application that is being evaluated is a part of. The evaluation score is identified by the foreign key evaluation id, which tells to what evaluation the score is for, the criteria name, which tells for what criteria it is applying to, and rubric id, to identify which rubric the criteria is from.