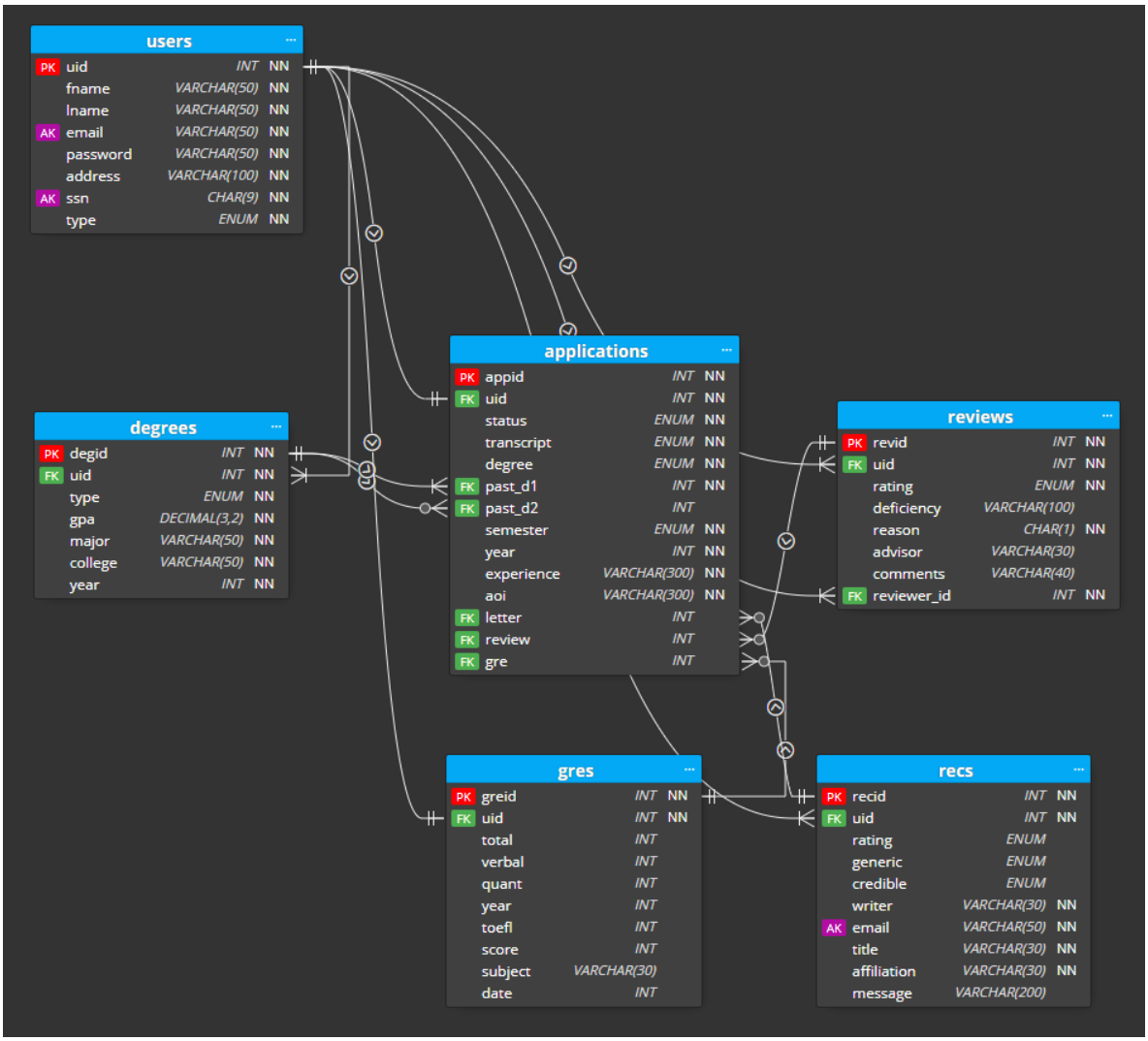
Phase 1 Project Report: Group 7 - Applications

Group Members: Benedek Sandor, Benjamin Chapman, and Rhonda Zakutney

SQL Table Diagram



Normal Forms:

Users: 1NF, because you can get the other attributes from only a partial key like (email) or (ssn), instead of the candidate key (uid, email, ssn)

Applications: 2NF, because multiple non-prime attributes depend on just the appid and the candidate key is (appid). If uid was unique it would be in 1NF as right now there can be multiple applications for the same user.

Degrees: 2NF, because there are no partial dependencies, degid and uid can identify all other attributes. But, it is not in 3NF because uid (a non-prime attribute) determines the non-prime attributes(type, major, year, college, gpa).

Gres: 1NF, because you can find the same information using partial key (greid) or (uid) from the candidate key (greid, uid)

Recs: 1NF, because email is unique so other attributes depend on a partial key(email) over the candidate key (recid, uid, email)

Reviews: 2NF, because multiple attributes depend on just the revid and the candidate key is (revid)

Assumptions made:

One assumption made was that the email for recommenders had to be unique, meaning that different applications cannot have the same recommender.

Missing functionality:

1. Refresh functionality that shows the most up-to-date database information. When two users are logged in at the same time and the database is updated, the new information is not shown on the other users end. Eg, when an applicant's application status is updated to complete the faculty reviewers page does not show the application upon refresh. It is only shown when the user logs out and logs back in.
2. CSS to make the pages more viewable/presentable.
3. Menu page for easy viewability on every user's page.
4. For some users the applicant's IDs are only shown and more information should be shown.

Work Breakdown

Benedek: I worked on the applicants, the recommenders bit, and the applicant create account. I did the form validation for the applications, creating an account, and implemented other security measures like bypass prevention. I also helped do a little bug fixing for the other members.

Ben: I worked on the Admin and GS page.

Rhonda: I worked on the pages for faculty reviewers and CAC, which also included creating the review form for the reviewer.

Short justification of key design choices:

Foreign key constraints were used to maintain referential integrity after considering the relationships between the tables. This design decision guarantees that data is consistent across the database. For instance, a user's former degrees, reviews, recommendation letters, and GRE scores are linked to an application via foreign keys. This makes it simple to query information about applications and enables the quick retrieval of associated data. The following are some of the specific instances foreign key constraints were used within our database:

- **applications** table links to the **users** table by the *uid* foreign key referencing the uid column from the users table, allowing each application to be linked to one specific user.
- use of the foreign keys *past_d1* and *past_d2* from **applications** table to connect to **degrees** table by referencing the *degid* column. This makes sure that each application is associated with the correct corresponding past degrees.
- *review* column from **applications** table is a foreign key referencing *revid* in **reviews** table, which makes sure that each application is linked to a valid corresponding review.
- *letter* column from **applications** is a foreign key referencing *recid* from **recs** table to make sure each application is linked to the correct recommendation later.

We also used data validation to make sure that the data stored within our database would be consistent. We had specific constraints like ‘NOT NULL’, ‘UNIQUE’, and ‘ENUM’ types for certain fields. These restrictions protect the accuracy of the data kept in the database by preventing the addition of invalid data. To ensure that only valid user roles can be stored, the type column in the user's table applied an ENUM constraint to either ‘'Admin', 'Applicant', 'GS', 'CAC', or 'Reviewer'. The type column in the user's table is what allowed us to differentiate between the different user roles within our system in order to ensure role-based access control so that each user would get access to features that were for their specific role. If we want to add new features later such as multiple departments or new application components, our schema would be able to support this just by adding new tables or columns with appropriate relationships made for existing tables.

For the web page key design choices, we made sure to divide the shown features based on user type and also use the database specifically to decide what to show. For example, the faculty reviewer would only be able to view the student’s application if the application is complete within our database. The reviewer would then be able to view the completed application and fill out the review form for the student. We also made sure in the recommender email section to allow our web page to only accept email addresses that were already existing as faculty within our database. This would prevent the user from entering an incorrect or invalid email address for the recommender email.