C. Logical Design

Student

Students(id, first_name, last_name, gender, DOB, matriculation, hsgpa, hsgrad, hsname, hsstate, hszip)

Student_degrees(student_id, degree_awarded, semester_awarded, year_awarded)

Reasoning: student_degrees has to be a separate table because a student may have multiple degrees. If there was only a Students table, that would lead to a lot of redundancy, since there could be many rows where the only thing different is the degree.

Faculty

Faculty(id, first_name, last_name, email, telephone, title, professor, status, title_type, highest_degree, discipline_highest_degree)

Faculty_departments(faculty_id, department_name)

Reasoning: faculty departments needs to be a separate table because it is a many to many relationship between faculty and departments. This table is shown in the ER diagram as the relationship "belongs to".

Departments

Departments(name, chair, room_number, building_code, telephone)

Reasoning: There is no reason to split this table further – already in BCNF.

Degree_programs

Degree_programs(name, acronym, director, department, credits, courses)

Coursework_requirements(program_name, course_acronym, course_num, reusable, min_grade)

Reasoning: coursework requirements is a weak entity contained within degree programs. Having it as a separate table fixes redundancy issues where all the fields in degree_programs would be the same, and the only difference would be the fields in coursework_requirements

Courses

Courses(acronym, num, description, program_name)

Prerequisites(prereq_acronym, prereq_num, course_acronym, course_num, min_grade)

Reasoning: it helps to have prerequisites as a separate table, since a course can have multiple prereqs. This makes it so there aren't a bunch of repeated courses, where the only difference is the prerequisite.

Course_offering

Course_offerings(course_acronym, course_num, section_num, semester_offered, year_offered, teacher)

Meeting_periods(course_acronym, course_num, section_num, semester_offered, year_offered, weekday, time_start, time_finish, room_number, building_code)

Reasoning: Meeting periods has to be a different table because there can be many of them per course. Start and finish time could change between meetings, as well as room and building, if there is some sort of discussion.

Relationships

Programs_pursuing(student_id, program_name, program_dropped)

Enrolled_courses(student_id, course_acronym, course_num, section_num, semester_offered, year_offered, grade_received)
Reasoning: programs pursuing links Students and Degree programs to show the majors each student is currently pursuing. This table is needed because Students to Degree programs is a many to many relationship, so there needs to be a third table to relate them. Enrolled courses relates Students to Course offerings. Again, this is a many to many relation, so there needs to be another table to relate the two entities.

D.
The design of the database is intentionally simple. Most of the constraints are handled through the interface, rather than through the database directly. The idea is that no users will directly be handling the database, rather, they will interact with it through the interface. This allows an admin to enter arbitrary values for testing directly into the database without worrying about whether they fit the constraints of the project. The only constraint in the database itself is foreign keys, which are necessary to keep all the data in line. Foreign keys are set up to cascade on delete.
Assumptions:

- credits are not listed anywhere in the requirements. It is assumed that each course is 3 credits.
- If a student drops a class, it is recorded as NG. If a student is currently taking a class and has not received a grade, it is recorded as NULL.
- A student may repeat a class 3 times, meaning he can take the class once initially, then repeat it 3 times (4 total attempts).

E.
The interface is written in python. MySQLdb must be installed (https://pypi.python.org/pypi/MySQL-python/1.2.4) . MySQLdb supports python 2.4 and 2.7, but not 3.0. The interface was tested on python 2.7.11. It implements the following constraints, as listed in the project requirements:

- Degree_programs – director must be full-time
- Departments – chair must be full-time, tenure-track; each department offers 1+ degree_programs
- Faculty_departments – each full-time faculty belongs to one department, part-time may belong to 1+
- Course_offerings – must have 1+ meeting_periods
- Enrolled_courses – students must satisfy 1+ prerequisite before enrolling; may repeat a course 3 times
- Students – have 1+ programs_pursuing

These constraints are implemented in the 'add' function. In addition, there is the assumption that all enrolled courses will be entered with a null grade_received. The grade can be changed using the 'edit' function.

F. User's manual

The python interface handles the constraints for the database. All interaction should occur through the interface for normal use. If there is a need for testing, the database can be accessed directly to bypass the constraints. The interface requires MySQLdb for python (see section E for more information).

The interface has four main functionalities: add, edit, delete, and query. It must be run using one (and only one) of the following flags:

Add is run with '-a' or '—add'. This function will prompt the user to select a table, and values for a row in that table. Constraints are checked at this time. If there is a violation, the program exits without changing the database. If not, the changes are saved.

Edit is run with '-e' or '—edit'. This function prompts the user to select a table, then select a field in that table to update, then the primary key of the row to update. Only fields that are not part of the primary key can be updated, as primary keys are meant to be permanent. If there is a desire to edit a field in the primary key, it is better to delete the row and add a new one.

Delete is run with '-d' or '—delete'. This function prompts the user to select a table, then select values for the primary key of the row he wants to delete. The row will be deleted, using foreign key cascade deletes on every row in other tables that depends on the initial row. In some cases, it is better to update first before deleting. For example, update the department chair before deleting the existing faculty id. This will prevent the delete to cascade, deleting the entire department, including all courses, course enrollments, and degrees.

Queries are run with '-qX', where X is replaced with the number of the query {1,2,…,10}. The queries are labelled in the order that they are presented in the requirements.

Misc.

The application is limited in that the interface does not list values in the tables (besides query 1). To view the full list of rows for each table, the database must directly be interacted with. The interface also performs only the queries listed in the requirements. There are many other queries that could be added, such as listing all students taking course X, listing all faculty of title Y, etc. The interface only has a single view as well. In reality, it should have separate views for students, teachers, admins, etc, so that a user can only edit relevant tables. It also follows the assumption that all courses are three credits (since credits are never listed in the requirements). This could be expanded later by adding in a credits field in the courses table.