1. The nouns are: faculty, students, courses, modules, lessons, order, widgets, topics, type, evaluation widgets, videos, slides, text documents, HTML, evaluations, essay, submission, exam, question, question type, course popularity, sections, semester, fall, spring, full summer, summer 1, summer 2, academic year, seat capacity, student type, enrollment, final grade, letter grade, student feedback, username, password, first name, last name, user, email, address, phone, benefit, tenure status, parking, bank account info, financial aid info, work-study, scholarship, GPA, grade, points lost, question worth.

“Nouns” were removed from sentences in which a certain user would “look” at a part of the application or think in certain ways on it. For example, the last two sentences describing what actions the student can take about their grades during instructor office hours has nothing to do with the application itself as long as it is providing the appropriate grades. It also does not matter if members of a certain class (such as Faculty) are able to modify the tables of another class (such as Course) because this does not imply a relationship between the two classes that require a foreign key. Furthermore, outside elements such as the “schedule” for changing the order of modules for the schedule have been removed, since this looks to be a manual operation that does not require an attribute in the database. There also should be added “name” attributes for most classes because otherwise manual identification becomes difficult. There were terms such as “student’s progress” for the exam that were also figurative (it does not actually track how far into the exam you are, but of the student’s own learning) and thus removed from the list of nouns. Unlike some other lists in the statement, the types of questions are not listed in the noun because I thought they could be brought under one umbrella attribute of “question type”.

1. The verbs are: contain, broken up into, rearranged, provide, build, come in, can be, evaluate, various types of questions, creates several, for a given, enroll in, sections for a course, register, users, multiple, neatly broken by, keeps track on how much.

Working off the nouns of the first part of the assignment, the verbs were specifically chosen to exist as relationships between those nouns I thought would exist as classes. Therefore, there are far more nouns than verbs, since many of my nouns are attributes and not classes. For example, although there exists the verb “author” that allows “faculty” to relate with “courses” since both “faculty” and “courses” would be defined as classes, there exists no verb to relate the noun “type” in class “widget.” If anything, the class “widget” would have the verb associated with it, but not the “type.” There also exist no verbs in which a certain user would CRUD on a certain database, since that has nothing to do with the table fields themselves.

1. Inheritance is specified by the arrowed lines. There exist clear cases where inheritance is necessary; for example, the term “User” exists in the original statement, so both “students” and “faculty” have to be under this umbrella term, and all three must be able to perform the actions a “user” would be able to take. Any case where a single type of object was singled out for further inspection and use could be considered for inheritance. Furthermore, the User was abstract because although both students and faculty are users, there cannot be an instance where a user is neither, as this is a learning application.
2. In general, the classes requiring other classes to bolster themselves requires aggregation or composition. For instance, if a learning module’s lessons could be enhanced using widgets, it would make sense to say that the lessons are composed of widgets, since it would not make sense for a lesson’s widgets to float around even after the lesson is deleted. I could not find any cases for aggregation, however, because whether if it’s questions for an exam, widget for a lesson, or module for a course, they would all have to be destroyed if the parent container is destroyed, since leaving these behind could lead to cheating and is otherwise unnecessary.
3. My naïve class diagram generally tried to stick with the idea of using “objects” as the classes and the “characteristics” as its attributes. For instance, there exists the “email” attribute under the “user” class because although “email” is a noun, it exists more clearly as a characteristic of “user”. I also tried to classify lists of how a certain class can be of certain types, such as how a widget can be a “video,” “slide,” or “evaluation widget,” into a single “type” attribute instead of creating separate specialized classes since not all of them require specialized fields for additional activity. As an added note, the difference between Widget’s type and Evaluation Widget’s eval\_type is that the Widget type specifies the type of Widget (video, slide, Evaluation Widget, etc.) and the eval\_type can be an essay, submission assignment, or exam. Needless to say, every case of “Evaluation Widget” would have its “type” set as “Evaluation Widget” and not “video” or “slides.”
4. The default data type was usually String. In cases where fields are required such as email or password, a limited varchar was used instead to make sure not too many characters are used. Integers were used in cases where only whole numbers would make sense, such as seat capacity, and floats were used for places where decimals would have to be necessary sometimes, as with GPA. Additionally, the data type for “popularity” under course is a float so you can rate out of 1.0, as a fraction of how many people enjoyed the class out of how many people were in it. Money, such as scholarship, was represented by a float. Date was used only to signify the year in Course, since certain courses (the statement said course, not section) were available only during certain years. Semester is an enumeration because there are only five types available for it. I later changed the data type for “Parking” under “Faculty” to Boolean from String because I realized that it should signify whether the faculty member has a permit to park within school grounds or not. To specify the “order” attributes in “Module” and “Lesson,” they are integers because they would presumably be set in an up-to-down list, and the “order” attribute would show how far they are from the top of the list.
5. In terms of cardinality, most relationships were defined by one-to-many relationships, with no many-to-many relationships as far as I could see. There was only one one-to-one relationship between “Student” and “Enrollment” that was necessary so that each student would be mapped to his or her experience with the course.
6. Although I initially thought that Modules would be linked to the Course as explained in the initial statement, I realized that professors and students would need to have section-specific modules since a course can have several different professors in different sections, leading to different assignments and evaluation widgets. I later found that Users can have multiple emails, addresses, and phones, and in order to normalize this, I created new tables for each. I deleted the “Assignment” class because I realized that it’s simply unnecessary since each instance of Evaluation Widget would act as the assignment for the student, and removed the “points\_lost” attribute because it gives essentially the same information as the “grade” attribute. I clarified the “Question” class by changing “points” to “available\_points” and adding a “correct” Boolean to keep track of whether the student got the question correct or not. Unfortunately, there can’t be an answer key to the questions because there cannot be a “correct answer” for something like an essay question.
7. I had no association classes and every idea that I had was able to be mapped onto my UML diagram. If I had to choose one, it would be the multiple email addresses, phone numbers, and addresses that would be linked to a single user. I could not have several fields to signify each because that would break first normal form, so I created a table for teach attribute with a one-to-many relationship so that a user can have as many of each as he or she wants. I also created an “Enrollment” class early on because I knew that there had to be an intermediary class between “Student” and “Section,” since a section would have multiple students but should also be able to hold a grade and student feedback for each student.