Project title: SQL gradebook

Project description: This project is intended to be an SQL database in which a professor can track the grades of each of their students. Prior to the beginning of each semester, the professor will obtain a list of students enrolled in each of their courses and this information will be stored in the database. Then, the professor will develop the various categories of assignments which their students will complete to earn a grade in the course (for example, one class may have homework, a project, and a final exam, while another may have only a midterm and a final) and the relative weights of those categories (for example, exams may be worth 40 percent of the grade, but homework is only worth 10 percent). Throughout the semester, each student will complete assignments to earn points toward their final grade. When a student completes an assignment, the professor will enter a score for that student's work in the database and the database will automatically place that score in the appropriate category (for example, if I am in a class that uses this gradebook system and I get 80 percent on exam 1, the professor would enter 80 for my score; then, based upon the pre-defined categories, the database would place that score in the appropriate category (I.E. exams)). At the end of the semester, a final grade will be calculated for each student. The final grade is equal to the weighted average over all category scores. The category score for a particular category is equal to the sum over all of the student's scores in that category divided by the maximum points that could be earned in that category. If a student hasn't completed any assignments in a particular category (for example, there haven't been any exams yet) the numeric grade would be computed as if that category didn't exist; if a student didn't earn a score for a particular assignment, that assignment wouldn't be factored in to the grade calculation for that student. This kind of calculation might be necessary if a student inquires about their progress in the course; the student could use this information in various ways (for example, to determine how much to study for an upcoming exam or to calculate what grade they need on an exam to achieve a given letter grade). However, prior to calculating a student's final grade, the professor should enter a score of 0 for any missing assignments. For example, if I earned 100/100 on the midterm and 85/100 on the final, my category score for the exams category would be (100+85)/(100+100) = 92.5 percent. If I earned a total of 200/200 for my homework, my category score for homework would be 100 percent. Finally, suppose I didn't complete the project; my category score for the project would be 0 percent. If exams are worth 50 percent, homework is worth 40 percent, and the project is worth 10 percent, my numeric grade would be (92.5%\*50+100%\*40+0%\*10)/(50+40+10)\*100 = 86.25%. The database will only calculate a numeric score (between 0 and 100 percent). The professor will then assign a letter grade (A, B, C, D, or F) according to their syllabus; this cannot be done in the database since every professor will have their own method of translating numeric scores into letter grades.

Team members: I will be the only person working on this project.

The following are the requirements for the SQL gradebook:

1. Information to be tracked:

The following is the information which the system should track:

Courses the professor teaches: Information about a course should include a unique identifier, a subject code (such as CS), a level (such as 3810), a section code (such as 3), a term (such as fall), and a year

Students who enroll in any of the professor's courses: Information about a student should include their 900 number (which should be unique) and the student's first and last name. Other information that the university might keep regarding a student, such as their home address, is not necessary for a gradebook system and should be excluded from the database.

A listing of (student, course) pairs so the professor can, for example, create a list of all students enrolled in a given course or create a list of all courses in which a given student is enrolled.

Assignment categories: In any of the professor's courses, the professor might give their students various kinds of assignments. For example, a course might have homework, exams, and a project. A list should be maintained of all categories defined by the professor in their various courses. Information about each category should include a unique identifier, the category description (such as homework), the course to which this category belongs, and the relative weight of that category in the grade calculation (for example, if the weight for exams is set to 50, the weight for homework is set to 40, and the weight for the project is set to 10, this would mean that exams account for 50 percent of the final grade, homework for 40 percent, and the project for 10 percent).

Assignments: A list should be maintained of all the professor's assignments. Information about the assignment should include a unique identifier, The assignment description (such as homework 1), the course to which this assignment belongs, the category within that course to which this assignment belongs, and the maximum number of points each student could earn on this assignment (excluding any extra credit the professor may authorize).

Scores: Finally, a list of all scores earned in the professor's various courses should be maintained. Information about each score should include the 900 number of the student who earned this score, the ID number of the course to which this score applies, the ID number of the assignment on which this score was earned, and the number of points earned (this is the raw number of points, such as 9, that a student has earned out of the number of points possible; the quotient of these numbers is the numeric score (usually between 0 and 1) upon which letter grades are based).

It should be possible to insert, delete, or update any of the above information (for example, the professor might mistype a score and wish to correct it or a student may graduate from the university and so the professor may wish to delete that student from the database to avoid unnecessary clutter)

Inserting, deleting, and updating each of these data objects (student, course, ETC) are distinct requirements with the following definitions:

Insert: To insert means to add a new record to the database. The record that is inserted should not already exist (for instance, it should not be possible to add multiple students with the same 900 number). Whenever a data object refers to another data object in the database, the system should insure that such object actually exists (for instance, it should not be possible to insert a score for an assignment that doesn't exist).

Update: to update a record means to modify a record that already exists in the database. It should be possible to modify any single characteristic of a given record. For example, it should be possible to modify a student's first name if they legally change their name from James to Jack. Such a change should not affect any other data about the student, such as what grades they earned on which assignments.

Delete: To delete a record means to completely remove that record from the database. The system should not allow a deletion if it would result in any data object referring to the deleted record. For example, suppose there is a homework category in a particular course and the professor has added several assignments to that category. The professor then decides that, since their students are performing well on their exams (despite not doing their homework), homework will no longer count toward the final grade. In this situation, the professor may wish to delete the homework category from the database. However, the system should require the professor to first delete all assignments in that category or move them to a different category before the deletion could occur. This prevents the situation where the system is computing a final grade, but one of the categories on which that grade is based no longer exists

2. Compute the percentage of points that a student has earned by completing assignments in a given category.

The percentage of points that a student has earned in a given category is equal to the sum of the scores earned in that category divided by the sum of the points possible for all assignments completed in that category. It is possible for such quotient to exceed 100 percent if the professor decides to award extra credit.

3. Compute the final grade for a specified student in a given course.

At the time when the professor wishes to assign final grades for a course, the system should compute the numeric score for each student. The numeric score is computed using the following algorithm:

1. If there are assignments in the course which a student has not completed, the system should fill in a score of 0 for that assignment. Incomplete assignments will be identified by comparing the list of assignments in the course with the list of assignments for which the student received a score. If there are any unscored assignments, the system should store a score of 0 for that assignment. A student may not have received a score for an assignment for several reasons; for instance, a student may have been absent during an exam or they may have failed to submit a homework assignment.

2. Then,for each category in the course, the system should compute the student's category score. A student's category score for a given category is equal to the fraction of the possible points that a student has earned in that category. For example, if the student earned 180 out of 200 possible points in a certain category, then their category score for that category is 0.9.

3. Then, the category score for each category derived in step 2 should be multiplied by that category's weight, which then yields the grade contribution for that category.

4. All such grade contributions derived in step 3 should be summed to yield the earned weighted score for that student.

5. Finally, the earned weighted score for a given student should then be divided by the sum of all weights to yield the final numeric score on which the professor will base letter grades. Usually, the professor will set the weights in such a way that they sum to 100; however, that constraint will not be enforced since the professor may wish to use a different system. The system will only calculate numeric grades; it is up to the professor to devise an algorithm for translating such scores into letter grades.

Business rules:

A Course may have many students and a student can enroll in several of the professor's courses.

A course should have at least one student (it doesn't make a lot of sense for a course to have no students; in that case, the university would likely cancel the course before it even began).

A student must enroll in at least one course (if a student isn't enrolled in any of the professor's courses, then the professor would not know of the existence of that student; in that case, the professor would not enter the student's information in the database).

No student may enroll in the same course twice in the same semester (for example, a student can't enroll in two instances of CS3810-002 in the same semester). I am including this rule only for completeness; the university's course registration system should prevent this situation from occurring. However, a student may repeat a course. For example, a student who fails CS3810 but wishes to use it to satisfy their degree requirements will repeat the course; for purposes of this database, this wouldn't be the same course because it would not have the same identifier.

A course may contain many assignment categories, but each assignment category belongs to only one course.

A course must have at least one category (if a course doesn't have any categories, then the grade calculations could not be performed; it is permissible for a course to only have one category. In that case, the weighted grade system is essentially reduced to a total points system).

An assignment category must be associated with exactly one course. This doesn't mean that if CS3810 has a final exam, no other course can have a final exam. It simply requires a method of distinguishing between these two categories.

A course may consist of many assignments, but each assignment is associated with only one course. This doesn't mean that the professor can't give students in more than one course the same assignment. If that occurs, then, for purposes of the grade calculation, these would be two distinct assignments. This setup is helpful because the professor could grade the assignment differently in each of their courses (for example, the professor may have higher standards for seniors than for juniors) or place the assignment in a different category (for example, the assignment could be homework for one course and a project for another course).

A category can have many assignments, but each assignment belongs to only one category. This means that an assignment cannot be counted as both homework and an exam in the same course. However, if the assignment is assigned in multiple courses, then each of those courses could place the assignment in a category appropriate for that course.

A course may yield many scores (a score is generated by the professor each time a student completes an assignment), but a score is associated with only one course (if a student's score could apply to multiple courses, then it will be extremely difficult to compute the student's final grade in a given course).

A student may complete many assignments (hopefully all of them), and an assignment may be completed by many students (hopefully all of them). A student might not complete any assignment (but then they would fail the course). An assignment might not be completed by any students (but then a zero score for that assignment will be used in the grade calculation).

A student may earn many scores (1 for each assignment they complete), but a score is earned by only one student. This doesn't mean that there can't be any group projects (as there often are in computer science courses). If there is a group project, the professor would enter a separate score for each student. This would allow the professor to award a lower grade to a student who didn't contribute to the group or to award a higher score to a group member whose contribution exceeded the professor's expectations.

An assignment may yield many scores (one for each student who completes it), but a score is associated with a single assignment. This doesn't mean that the professor can't give multiple students the same score on the same assignment; however, it is necessary to develop a method of determining which scores apply to which students.

erd:

The relationship "student enrolls in course" is m:m. The cardinality for both student and course is (1,n).

The relationship "course contains category" is 1:m. The cardinality for category is (1,n). The cardinality for course is (1,1).

The relationship "course consists of assignment" is 1:m. The cardinality for assignment is (1,n). The cardinality for course is (1,1).

The relationship "category has assignment" is 1:m. The cardinality for assignment is (1,n). The cardinality for category is (1,1).

The relationship "course yields score" is 1:m. The cardinality for score is (0,n) (there will be no score if all students fail to complete any assignments). The cardinality for course is (1,1).

The relationship "student completes assignment" is m:m. The cardinality for both student and assignment is (0,n).

The relationship "student earns score" is 1:m. The cardinality for score is (0,n) (if a student doesn't complete any assignment, then they will not earn any scores). The cardinality for student is (1,1).

The relationship "assignment yields score" is 1:m. The cardinality for score is (0,n) (if no student completes a given assignment, then there will be no scores associated with that assignment). The cardinality for assignment is (1,1).

The relationship "student generates enrollment" is 1:m. The cardinality for enrollment is (0,n) (A student might not enroll in any classes. However, in that case, they likely wouldn't be in the database unless they dropped all of their classes later in the semester). The cardinality for student is (1,1).

The relationship "course yields enrollment" is 1:m. The cardinality for enrollment is (0,n) (this allows for a situation in which there are no students enrolled in a course; it is included only for completeness. If there were really a course in which no students enrolled, the university would likely cancel the course before it even began). The cardinality for course is (1,1).

The relationship "enrollment yields attempt" is 1:m. The cardinality for attempt is (0,n) (A student might not attempt any assignments in a given course). The cardinality for enrollment is (1,1).

The relationship "assignment generates attempt" is 1:m. The cardinality for attempt is (0,n) (An assignment might not be attempted by any student, although this is unlikely to occur). The cardinality for assignment is (1,1).

The relationship "attempt yields score" is 1:1 (each student will earn exactly one score for each assignment they complete. If a student could earn multiple scores for the same assignment, then the grade calculations will be extremely difficult).

Course has attributes course\_id (pk), course\_subject, course\_level, course\_section, course\_description, course\_term, and course\_year

Student has attributes student\_id(pk), student\_fname, student\_lname, student\_initial, and student\_suffix.

Enrollment has attributes enrollment\_id(pk), student\_id(fk), and course\_id(fk).

Category has attributes category\_id(pk), category\_description, course\_id(fk), and category\_weight.

Assignment has attributes assignment\_id(pk), assignment\_description, category\_id(fk), and assignment\_points\_possible.

Attempt has attributes attempt\_id(pk), enrollment\_id(fk), and assignment\_id(fk).

Score has attributes score\_id(pk), attempt\_id(fk), and score\_points\_earned.

Design for the SQL gradebook

Tables: the following are the tables which will be used in the sql database:

Course: the course table tracks all courses the professor teaches. This table has the following attributes:

1. course\_id: this attribute is an integer that uniquely identifies this course and is the primary key for this table. The table will be created such that, whenever a new course is added, the course\_id is one larger than the course\_id of the most recently added course. For example, if the last course\_id used was 12, the course\_id for the next course to be added will be 13.

2. Course\_subject: this is a 2 or 3-letter subject designation that determines the general subject matter covered in this course. For example, computer science courses have a course\_subject of CS. Each course must have a subject designation.

3. course\_level: this is an integer that determines the difficulty level of the course. For example, courses with a course\_level between 3000 and 3999 are junior-level courses. Every course must have a level and the level should not exceed 9999.

4. course\_section: During any given semester, multiple sections of the same course may be offered. For example, section 1 of CS3810 may be offered on tuesday and thursday, whereas section 2 may be offered on monday and wednesday. The course\_section attribute is an integer that identifies the specific offering of a course. For example, CS3810-002 will have a course\_section of 2. Every course must have a section and the course\_section should not exceed 99.

5. Course\_desc: this is a string that contains the course description. For example, the course\_desc for CS3810 might be "principles of database systems". Every course must have a description.

6. Course\_term: the course\_term indicates when, during a given calendar year, a particular course is offered. For example, the term for this semester is fall. Every course must have a term and the valid entries for course\_term are fall, spring, and summer.

7. course\_year: this is the calendar year in which the course is offered. Every course must have a year and the year should not exceed 9999.

Student: the student table will track all students who enroll in any of the professor's courses. The table will have the folloing attributes:

1. student\_id: this is an integer that uniquely identifies a given student and is the primary key for this table. At MSU denver, the 900 number is used for this purpose. The professor may elect to use a different identification scheme; however, the database should ensure that no two students can have the same student\_id. Every student must have an id number and this number should not exceed 999999999.

2. student\_fname: This is the student's first name. Every student must have a first name.

3. student\_lname: this is the student's last name. Every student must have a last name.

4. student\_initial: this is the student's middle initial. A student might not have a middle initial.

5. Student\_suffix: this is a suffix, such as jr., that may be included in some names. A student might not have a suffix.

Enrollment: this table lists courses in which a given student is enrolled. The enrollment table has the following attributes:

1. enroll\_id: this is an integer which uniquely identifies this enrollment and is the primary key for this table.

2. Student\_id: this attribute is a foreign key and indicates the student to which this enrollment pertains. This attribute is required.

3. course\_id: this is a foreign key and indicates one course in which the student is enrolled. This field is required.

The combination of student\_id and course\_id should be unique. However, a surrogate key is being used here to prevent the situation in which the professor enters an assignment score for a student who is not enrolled in a given course. Upon deleting a student (for example, because they graduated from the university), all enrollments in which the student\_id matches the student\_id of the deleted student should be deleted. Upon deleting a course (for example, because it was canceled), all enrollments in which the course\_id matches the deleted course should be deleted. Whenever either the course\_id or the student\_id is updated, all references to that course\_id or student\_id should be updated.

Categories: this table tracks the various categories of assignments the professor might give to their students (for example, common categories include homework, exams, and projects). The category table has the following attributes:

1. cat\_id: this is the primary key for this table. The id should be assigned in such a way that it is always one larger than the previously used id number.

2. cat\_desc: this attribute is the category's description. For example, common values of this attribute might be "homework" or "final exam".

3. course\_id: this attribute is a foreign key that indicates the course to which this category belongs. This attribute is required. Upon deleting a course, all categories in that course will be deleted If the course\_id is updated, then all references to that course\_id should also be updated.

4. cat\_weight: this attribute indicates the weight this category is to be given in any grade calculations involving it. The weight should always be positive. This attribute is required. Note that the weight has no particular units; it is the relative magnitude of the weight that determines how much this category contributes to a student's final grade. For example, suppose a course has two categories and that category a is worth 80 percent of the grade and category b is worth 20 percent. Then, for any positive real number x, the weights for categories a and b can be set to 80\*x and 20\*x respectively; the grade calculation will be identical regardless of the value of x.

Assignment: this table tracks all assignments given by the professor. The assignment table has the following attributes:

1. asg\_id: this is the primary key that uniquely identifies this assignment. The value of this field should always be one larger than the value for the most recently added assignment.

2. Asg\_desc: this is the description of the assignment. Sample values of this attribute are "homework 1" and "final exam". This attribute is required.

3. cat\_id: this is a foreign key that indicates the category to which this assignment belongs. Note that the course\_id will not be included as originally specified because that would create an undesirable transitive dependency (course\_id depends on cat\_id which depends on asg\_id). This attribute is required. Upon deleting a category, all assignments in that category are also deleted; this is done to prevent the situation where an assignment could belong to a non-existent category. Upon updating the cat\_id attribute in the category table, the cat\_id attribute in this table will also be updated.

4. asg\_points\_possible: this is the maximum number of points that a student could earn on this assignment (excluding any extra credit). This attribute should be a positive integer. This attribute is required.

attempt: this table will track which assignments were completed by each student in a given course. This table has the following attributes:

1. Att\_id: this attribute is the primary key that will uniquely identify this attempt. This attribute should have a value that is one larger than the value for the most recently added attempt.

2. Enrollment\_id: this attribute is a foreign key that indicate the enrollment to which this attempt applies. This attribute is required. Upon deleting an enrollment (for example, because a student dropped the course), all attempts associated with that enrollment are also deleted. If the enrollment\_id attribute in the enrollment table is updated, then all references to that enrollment\_id in this table should be updated in the same manner.

3. asg\_id: this is the assignment that was attempted by a particular student in a given course (the student and course are determined by the enrollment\_id). This attribute is a foreign key and is required. Upon deleting an assignment from the assignment table, all attempts associated with that assignment are also deleted. If the asg\_id is updated in the assignment table, then all references to that assg\_id are updated in the same manner.

The combination of enrollment\_id and assg\_id must be unique.

Score: this table tracks the scores earned on each of the professor's assignments. This table has the following attributes:

1. score\_id: this is the primary key that uniquely identifies this score. This attribute should have a value that is one greater than the value for the most recently added score.

2. att\_id: this is a foreign key that indicates the attempt to which this score applies. This attribute is required. Upon deleting an attempt, any score associated with that attempt is deleted.

3. score\_earned: this is the number of points, relative to the number of points possible, that have been earned on this attempt. For example, if a student earns 9 points out of 10 on an assignment, then 9 is entered for this attribute. This attribute should be a non-negative number and is required. Note that this attribute may exceed the points possible for the corresponding assignment if the professor awards extra credit.

Att\_id must be unique since a student can't earn multiple scores on the same assignment in the same course (for example, a student earns a single score for the final project in CS 3810).

Functionality:

1. the system should attempt to connect to the database. If the connection fails, then an error message is displayed and the system exits. If the connection succeeds, then the system continues on to step 2.

2. The system should display a menu of all available actions. Possible options are:

a. Update, insert, and delete actions for course, student, enrollment, assignment, category, and score.

b. calculate current grade for a given student and course.

c. calculate final grade for a given student and course.

d. quit.

3. When a valid action is chosen, the appropriate procedure should be carried out to perform that action (for example, if the professor chooses "update student", then the system should ask for a student's name to enter into the database).

4. Steps 2 and 3 should be repeated until the professor chooses the quit action; when this occurs, the database connection will be closed and the system will exit.

The available actions are defined as follows:

Insert student: if the professor selects the insert student action, the system will ask for a 900 number, a first name, a last name, a middle initial, and a suffix. The system will then issue the following query to the database to insert the student record: insert into student values(<stu\_id>,<stu\_fname>,<stu\_lname>,[stu\_mi],[stu\_suffix]); where stu\_fname is the first name, stu\_lname is the last name, and, if the professor enters a non-empty string for stu\_mi and stu\_suffix, these values represent the stu\_mi and stu\_suffix attributes respectively. If a student with the 900 number entered by the professor already exists, then the system should issue an error message of the form "student <stu\_id> already exists". If the professor enters an empty string for the middle initial or suffix, these fields are set to null.

delete student: if the professor selects the delete student action, the system should generate a list of all students who exist in the database. Then, the professor should enter the 900 number of the student they wish to delete. If the professor enters a valid id number, the system should issue the following query to the database to delete the student record:

delete from student where stu\_id = <stu\_id>;

where <stu\_id> is the id of the student who is to be deleted. If the professor enters an invalid id number, the system issues an error message of the form "student <stu\_id> does not exist" where <stu\_id> is the invalid student id number.

update student: If the professor selects the update student action, then the system will generate a list of all student records. The professor will then enter the 900 number of the student they wish to update. If the professor enters a 900 number that doesn't exist, then the system issues an error of the form "student <stu\_id> does not exist". Otherwise, the system asks which attribute the professor wishes to update. If the professor enters a valid attribute name, the system asks for the new value of the attribute, which must be of the correct data type. If a valid attribute name and value were entered, then the system issues a query of the form update student set <attribute\_name> =<attribute\_value> where stu\_id = <stu\_id>;.

Insert course: if the professor selects the insert course action, the system will ask for a course\_subject, a course\_level, a a course\_section, a course\_term, and a course\_year. The system will then issue the following query to the database to insert the course record: insert into course(<course\_subject>, <course\_level>, <course\_section>,<course\_term>,<course\_year>) values(<course\_subject>,<course\_level>,<<course\_section>,<course\_term>,<course\_year>);. If any of these attributes has an invalid value, then the system should not insert the record but instead will issue an error of the form "invalid value <attribute\_value> for <attribute\_name>".

delete course: if the professor selects the delete course action, the system generates a list of all courses. The professor then enters the id of the one they wish to delete. The system then issues a query of the form delete from course where course\_id = <course\_id>;. If the course id exists, the corresponding record is deleted; otherwise, the system issues an error of the form "course <course\_id> does not exist".

update course: if the professor selects the update course action, the system generates a list of all courses. The professor then enters the id number of the course to be updated. The system then generates a list of all attributes for the course, excluding the id, which cannot be updated. The professor then enters the name of the attribute they wish to update followed tby its new value. If either the attribute name or value are invalid, the system issues an appropriate error message; otherwise, the course record is updated.

Insert enrollment: If the professor selects the insert enrollment action, then the system will ask for a student id and the course in which this student is enrolled. If either the student id or the course id does not exist, the system should issue an appropriate error message. Otherwise, the system should issue the appropriate sql query to insert the enrollment record.

Delete enrollment: if the professor selects the delete enrollment action, the system generates a list of all existing enrollments. The professor then enters the id number of the enrollment they wish to delete. If the enrollment id doesn't exist, then the system issues the appropriate error message. Otherwise, the appropriate sql query is issued to delete the enrollment record indicated by the entered enrollment id.

update enrollment: if the professor selects the update enrollment action, then the system generates a list of all existing enrollment records. The professor indicates which enrollment is to be updated by entering its id number. Then, a list of modifyable attributes is generated and the professor then enters the name of the attribute that is to be modified and its new value. If any value doesn't exist or is invalid, the system issues an appropriate error message. Otherwise, the appropriate sql query is issued to update the given attribute to its new value.

Insert category: If the professor selects the insert category action, then the system will ask for a description of the category (such as homework), the course to which this category belongs, and the relative weight of that category within that course. If any of the supplied values do not exist or invalid, the system issues an appropriate error message. Otherwise, a this category will be added to the database.

Delete category: if the professor selects the delete category action, then the system will generate a list of all existing categories and the professor then enters the id number of the category they wish to delete. If the specified category doesn't exist, the system issues an appropriate error message. Otherwise, the specified category is deleted.

Update category: If the professor selects the update category action, then the system will generate a list of all existing categories. Then, the system will ask for the id number of the category to be updated. Then, the system will ask for the attribute (such as the the category description) within that category to be updated and the new value of that attribute. If any of the supplied values does not exist or is invalid, the system issues an appropriate error message. Otherwise, the specified category is modified to reflect the new attribute value.

Insert assignment: If the professor selects the insert assignment action, then the system will ask for an assignment description (for example, homework 1), the category to which this assignment belongs (selected from all existing categories), and the number of points possible on this assignment. If either the category id or the points possible are invalid (for example, the professor enters a negative value for points possible), then the system will issue an appropriate error message. Otherwise, the assignment record will be added.

Delete assignment: if the professor selects the delete assignment action, then the system asks for the id number of the assignment to be deleted (selected from all existing assignments). If the professor enters an id number that doesn't exist, then the system issues an appropriate error message. Otherwise, the assignment record is deleted.

Update assignment: If the professor selects the update assignment action, then the system will ask for the id number of the assignment to be updated (selected from all existing assignments), the attribute within that assignment (selected from all available attributes) to be modified, and the new value of that attribute. If any of the supplied values does not exist or is invalid, the system issues an appropriate error message. Otherwise, the assignment record is updated.

insert score: If the professor selects the insert score action, then the system will ask for the course, student, and assignment to which this score applies (selected from all records in their respective tables) and the number of points earned on the assignment. If any of the supplied values does not exist or is invalid, an appropriate error message is issued; otherwise, the assignment score is inserted into the database.

Delete score: If the professor selects the delete score action, then the system asks for the id number of the score to be deleted (selected from all existing scores). If the id number is invalid, the system issues an appropriate error message. Otherwise, the score is deleted.

Update score: if the professor selects the update score action, then the system asks for the score to be modified (selected from all existing scores), the attribute within that score to be modified (selected from all available attributes), and the new value of that attribute. If any of the supplied values does not exist or is invalid, then the system issues an appropriate error message. Otherwise, the score record is updated.

Finally, the system should be able to calculate the current or final course grade for a given student as follows:

1. If there are assignments in the course which a student has not completed, the system should fill in a score of 0 for that assignment. Incomplete assignments will be identified by comparing the list of assignments in the course with the list of assignments for which the student received a score. If there are any unscored assignments, the system should store a score of 0 for that assignment. A student may not have received a score for an assignment for several reasons; for instance, a student may have been absent during an exam or they may have failed to submit a homework assignment. This step doesn't apply for calculating a current course grade; if there are any unscored assignments, they are simply ignored.

2. Then,for each category in the course, the system should compute the student's category score. A student's category score for a given category is equal to the fraction of the possible points that a student has earned in that category. For example, if the student earned 180 out of 200 possible points in a certain category, then their category score for that category is 0.9. When calculating current course grades, the system should ignore any category in which no scores have been entered to prevent division by 0 errors (this may occur if there haven't been any tests, for example).

3. Then, the category score for each category derived in step 2 should be multiplied by that category's weight, which then yields the grade contribution for that category.

4. All such grade contributions derived in step 3 should be summed to yield the earned weighted score for that student.

5. Finally, the earned weighted score for a given student should then be divided by the sum of all weights to yield the final numeric score on which the professor will base letter grades. When calculating current course grades, the sum of all weights in this calculation is replaced with the sum of all weights for categories in which scores have been entered. Usually, the professor will set the weights in such a way that they sum to 100; however, that constraint will not be enforced since the professor may wish to use a different system. The system will only calculate numeric grades; it is up to the professor to devise an algorithm for translating such scores into letter grades.