1. Project Description

The project presents a complete solution to connect to a MySQL server with user defined settings and it can establish the Student Management System. The software developed in this project can create its own database and it can populate the database with random but meaningful data for testing or evaluation purposes. The database stores student, professors, course and classes and student clubs, and how these relate to each other. For example, the database can help student find out the average grades from previous years and take the course of their interest in a semester where they have a suitable workload, this would help students plan their semesters better and become more successful. The students can also see how professors grade the previous students and adjust their effort so that they achieve their expectations. Obviously, this student management system like all the existing solutions inform the students about their current grades, their performance in each class, the credit hours they have passed and all the credit hours they have taken. Moreover, the school administration can track all the student clubs, students involved with the clubs, and their level of involvement. This could be especially helpful for schools when they want to perform contact tracing in the days of COVID-19, since the students are very likely to spend time with other students in the clubs and with this tool they can easily see who is possibly affected and who is not. This Student Management System has an easy interface for people who are not tech savvy. Familiarity with Microsoft Windows should be enough for most users to get the most benefits out of all the data at their fingertip. Overall, Student Management System is a necessary tool for the schools, to keep track of all the information in a secure and safe manner.

While using a database for school's information has many benefits, it is important to compare the database with more traditional solutions. Unlike paper which has served the humanity for many years, databases do not result in paper waste and are always accessible to multiple people without a need for duplication. These results in a greener world and obviously a more efficient working environment where data is available to all members of the school, so that they can make the best decisions. Also, unlike paper once the data becomes obsolete, removing information from databases is significantly easier than taking care of many paper documents containing personal information. To make it even easier, in this software you can remove information by either typing them one by one if something very specific must be deleted or you can use an interactive table to change larger portion of the information. You can as easily write or modify the tables so that the database remains updated, which is one of the major problems with more traditional solution, where normal users had to go through many complex steps to edit a single cell in a database.

If you are in doubt, this software has the capability to generate data in different sizes and magnitudes. For an accurate test you only need several numbers close to actual numbers in the school of your interest, in just few second you can test all the features and experience the latency with this database software. Overall, I hope this solution evolve over time to help students make wiser decisions about their education and future life. There is still many more features that can be added however they would require more time and care especially on the GUI end. SQL quires are very capable and very efficient, but on the GUI end many hours must be spent to make the program look better and easier to use.

2. E/R Diagram

In this E/R diagram we have 5 entity sets and 6 relations for the student system proposed. The entity sets are 1) Students, 2) Professor, 3) Department, 4) Class, 5) Course, and 6) Club. The relations are 1) Offer, 2) Subject, 3) StudentDepartment, 4) Teach, 5) Taken, 6) Member, and 7) President. Each of these entities and relationships are described below:

Entity Sets

1) Students: each entry represents a student with a unique UIN followed by his/her name, birthdate, phone number and email address. In terms of data types UIN is a number, birthdate is a date and all other ones are string. Phone number is string to hold 0 (at the beginning) or + if needed for country codes or other types of code. The key for this set is UIN (“uin”) and it is a number for better performance. The assumption is that a student may be given a new UIN if he/she register in the university again and he/she could have the same email and phone number, as a result they would not be unique and cannot be a key. Moreover, since phone numbers get reassigned, the phone number attribute cannot be a key.
2) Professor: each entry represents a professor with a unique Professor ID followed by his/her name, birthdate, phone number and email address. In terms of data types Professor ID is a number, birthdate is a date and all other ones are string. Phone number is string to hold 0 (at the beginning) or + if needed for country codes or other types of code. The key for this set is Professor ID (“profId”) and it is a number for better performance. The assumption is that a professor may be given a new profId if he/she get employed by the university again and he/she could have the same email and phone number, as a result they would not be unique and cannot be a key. Moreover, since phone numbers get reassigned, the phone numbers attribute cannot be the key.
3) Department: each entry represents a department with a unique Department ID followed by department’s name, phone number and email address. In terms of data types Department ID is a number and all other ones are string. Phone number is string to hold 0 (at the beginning) or + if needed for country codes or other types of code. The key for this set is Department ID (“depId”) and it is a number for better performance; “email” and “name” are keys too. The assumption is that an email is not reassigned to a different department and if departments are merged or separated, they would be given new emails or only one of them use the old email. As for the phone number there is a high likely hood that a phone number gets shared among small departments, that is why it cannot be a key. The departments name is unique at any given time, but the department may change its name and that is why it is not selected as the key. This way the relations “offer” and “StudentDepartment” can stay consistent when a department changes its name.

4) Class: each entry represents a class with a unique Class ID followed by year and semester. In terms of data types Class ID, year and semester are a number. The key for this set is Class ID (“classId”). The classes are each given a unique id (“classId"), this is done so that it is not needed to store the year and the semester for each entity in the relations “Teach” and “Taken” and instead using the unique code which stays constant. Semester is a number, either 1, 2 or 3. 1 being Fall, 2 being Spring, and 3 being summer. Take note that class stores Class ID, year, and semester only, but it does not store the Course Code, name or number of credits. This is done so that the course information does not have to repeat for repeating classes every year/semester.
5) Course: each entry represents a course with a unique Course Code followed by name and credit. In terms of data types Course Code and name are string, but credit is a number. The key for this set is Course Code (“courseCode”), this is the only set with a string key and this is done to compare performance. The courses are each given a unique code (“courseCode"), this is done because different department can offer courses with same name but different codes to distinguish them from one another. Take note that Course stores courseCode, name and credit only, but it does not store the year or semester. This is done so that the course information does not have to repeat for repeating classes every year/semester.
6) Club: each entry represents a club with a unique Club ID followed by name, and email. In terms of data types Course ID is a number and all other ones are string. The key for this set is Club ID (“clubId”). “clubId” is the number assigned to each club, the clubs may change their name and while their names must be unique, they can be changed unlike the “clubId”, that is the reason why “clubId” is selected as the key. The same applies to the email, it can be changed but at any given time the email must be unique. This allows the name change to happen without a need to modify the relations “President” and “Member”, also this would allow consistency in case a club changed its name or email.

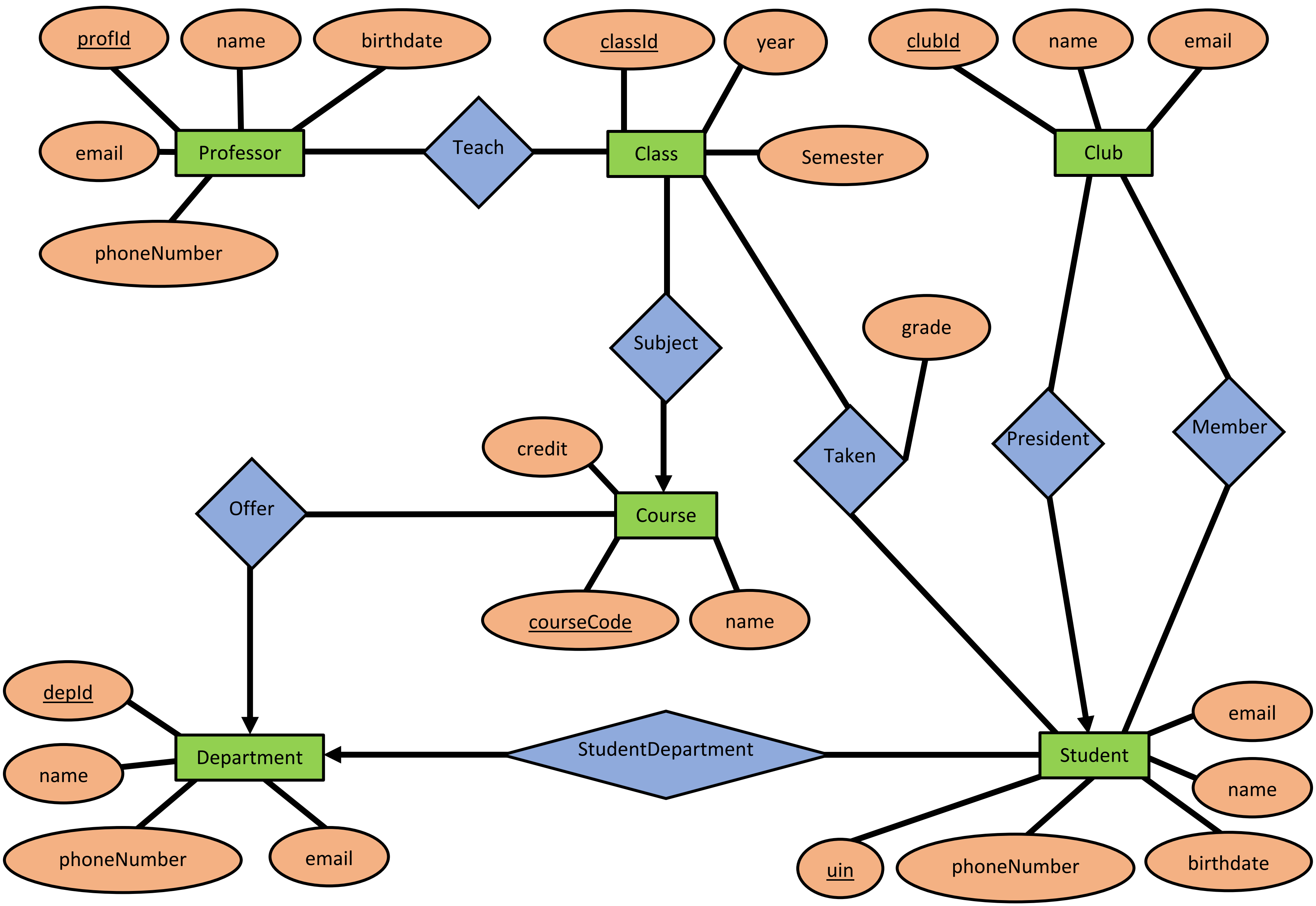
Relationships:

1) Offer: This is the relationship between each department and each course, this allows us to know which department is offering which course. For every course there is one department but obviously a department can offer many courses. So, this is a many to one relationship.
2) Subject: This is the relationship between each class and the course being taught, this allows us to know which subject is taught in a class. For every class there is one course but obviously a subject can be taught in more than one class. So, this is a many to one relationship.
3) StudentDepartment: This is the relationship between each department and each student, this allows us to know each student’s department. For every student there is one

department but obviously a department can have many students. So, this is a many to one relationship.

4) Teach: This is the relationship between the professors and the classes, this allows us to know the professors for each class and classes each professor teaches. Each professor can teach more than one class and each class can be taught by more than one professor. So, this is a many to many relationship.
5) Taken: This is the relationship between students and the classes he/she has taken, this allows us to know the classes taken by each student and all students in each class. Each student can have more than one course and each course can have many students. So, this is a many to many relationship. For each student taking a class, he/she gets a grade and the grade is stored as an attribute in the relation “taken”. Take note that this is only for classes which are Taken so all of them should have a grade and grade is a number.
6) Member: This is the relationship between the students and the clubs they are member of, this allows us to know the students for each clubs and all students in each club. Each student can be ma member of more than one club and each club can have many members. So, this is a many to many relationship.
7) President: This is the relationship between clubs and the president, this allows us to know which student is the club president. For every club there is one president but each student can be president of more than one club. So, this is a many to one relationship.

Alireza Safdari Gandhari Project 1 10/20/2020



3. Converting E/R Diagram to Database Schema

Relations are:

For Entity Set

1) Student(uin, name, birthdate, phoneNumber, email, depId)
2) Professor(profId, name, birthdate, phoneNumber, email)
3) Department(depId, name, phoneNumber, email)
4) Class(classId, year, semester, courseCode)
5) Course(courseCode, name, credit, depId)
6) Club(clubId, name, email)

Relation for relationships:

1) Teach(profId, classId): many to many
2) Taken(uin, classId, grade): many to many
3) Member(uin, clubId): many to many
4) President(clubId, uin): many to one, while this can be combined in Student, it would produces many NULLs in that column since the number of clubs to number of students is very small and each club has only one president (since it many to one relationship). So it would be better not to combine this relationship.
5) Offer(courseCode, depId): many to one, combined in Course
6) Subject(classId, courseCode): many to one, combined in Class
7) StudentDepartment(uin, depId): many to one, combined in Student

Nontrivial Functional Dependencies:

Entity Sets:

1) In Student: $\{uin \rightarrow \{name, birthdate, phoneNumber, email, depId\}\}$
2) In Professor: $\{profId \rightarrow \{name, birthdate, phoneNumber, email\}\}$
3) In Department:$\{depId \rightarrow \{name, phoneNumber, email\}, email \rightarrow \{depId, name, phoneNumber\}, name \rightarrow \{depId, phoneNumber, email\}\}$
4) In Class:$\{classId \rightarrow year, semester, courseCode\}$
5) In Course:$\{courseCode \rightarrow name, credit, depId\}$
6) In Club: $\{clubId \rightarrow \{name, email\}, name \rightarrow \{clubId, email\}, email \rightarrow \{clubId, name\}\}$

Relationships:

1) In Teach: $none$
2) In Taken:$\{uin, classId \rightarrow grade\}$
3) In Member: $none$
4) In President: $\{clubId \rightarrow uin\}$

Boyce-Codd Normal Form (BCNF) normalization:

In order to perform BCNF normalization, few steps must be followed.

a. The keys should be found
b. Violation must be identified. Any functional dependency that its left side is not a superkey is a violation.
c. If a violation was found, take the closure of the left side and create a new relation with the attributes in closure result, and another relation with all attributes left in the original relation followed by the left side of the functional dependency in violation. Now that there are two relations, make sure each has the associated functional dependencies (where both left and right side coexist in one relation).
d. Go back to step b and repeat till no functional dependency is in violation of BCNF (functional dependencies from all the relations generated from the original relation must be checked)

Now that the normalization process is laid out, this process is applied to each relation.

Entity Sets:

1) Student(<u>uin</u>, name, birthdate, phoneNumber, email, depId) with $\{uin \rightarrow \{name, birthdate, phoneNumber, email, depId\}\}$
   a. The key for this set is uin, because $uin^+ = \{uin, name, birthdate, phoneNumber, email, depId\}$. uin is the only key.
   b. Since the left side of the only functional dependency is a key and a key is always a superkey, there is no BCNF violation.
   c. N/A
   d. N/A
2) Professor(profId, name, birthdate, phoneNumber, email) with $\{profId \rightarrow \{name, birthdate, phoneNumber, email\}\}$
   a. The key for this set is $profId$, because $profId^+ = \{profId, name, birthdate, phoneNumber, email\}$. $profId$ is the only key.
   b. Since the left side of the only functional dependency is a key and a key is always a superkey, there is no BCNF violation.
   c. N/A
   d. N/A
3) Department(<u>depId</u>, name, phoneNumber, email) with $\{\mathrm{depId} \rightarrow \{name, \mathrm{phoneNumber, email}\}, \mathrm{email} \rightarrow \{\mathrm{depId}, name, \mathrm{phoneNumber}\}, name \rightarrow \{\mathrm{depId, phoneNumber, email}\}\}$
   a. The key for this set is $depId$, because $depId^+ = \{depId, name, phoneNumber, email\}$. $depId$ is not the only key. email and $name$ are the other 2 keys available in this relation, since closure of both

would give all othe attributes. $email^+ = \{depId, name, phoneNumber, email\}$ and $name^+ = \{depId, name, phoneNumber, email\}$

b. Since the left side of the all three functional dependency is a key and a key is always a superkey, there is no BCNF violation.
c. N/A
d. N/A

4) Class(<u>classId</u>, year, semester, courseCode) with $\{classId \to year, semester, courseCode\}$
   a. The key for this set is $classId$, because $classId^+ = \{classId, year, semester, courseCode\}$. $classId$ is the only key.
   b. Since the left side of the only functional dependency is a key and a key is always a superkey, there is no BCNF violation.
   c. N/A
   d. N/A
5) Course(<u>courseCode</u>, name, credit, depId) with $\{courseCode \to name, credit, depId\}$
   a. The key for this set is $courseCode$, because $courseCode^+ = \{courseCode, name, credit, depId\}$. $courseCode$ is the only key.
   b. Since the left side of the only functional dependency is a key and a key is always a superkey, there is no BCNF violation.
   c. N/A
   d. N/A
6) Club(<u>clubId</u>, name, email) with $\{clubId \to \{name, email\}, name \to \{clubId, email\}, email \to \{clubId, name\}\}$
   a. The key for this set is $clubId$, because $clubId^+ = \{clubId, name, email\}$. $clubId$ is not the only key. $name$ and email are the other 2 keys available in this relation, since closure of both would give all othe attributes. $email^+ = \{clubId, name, email\}$ and $name^+ = \{clubId, name, email\}$
   b. Since the left side of the all three functional dependency is a key and a key is always a superkey, there is no BCNF violation.
   c. N/A
   d. N/A

Relationships:

1) Teach(profId, classId) with no nontrivial functional dependency
   a. The key for this set is $\{profId, classId\}$.
   b. Since there is no functional dependency, there is no BCNF violation.
   c. N/A
   d. N/A
2) Taken(uin, classId, grade) with $\{uin, classId \to grade\}$

a. The key for this set is $\{uin, classId\}$, because $\{uin, classId\}^+ = \{uin, classId, grade\}$, $\{uin\}^+ = \{uin\}$, and $\{classId\}^+ = \{classId\}$. $\{uin, classId\}$ is the only key.
b. Since the left side of the only functional dependency is a key and a key is always a superkey, there is no BCNF violation.
c. N/A
d. N/A

3) Member(uin, clubId) with no nontrivial functional dependency
   a. The key for this set is $\{uin, clubId\}$.
   b. Since there is no functional dependency, there is no BCNF violation.
   c. N/A
   d. N/A
4) President(<u>clubId</u>, uin) with $\{clubId \rightarrow uin\}$
   a. The key for this set is $clubId$, because $clubId^+ = \{clubId, uin\}$. $clubId$ is the only key.
   b. Since the left side of the only functional dependency is a key and a key is always a superkey, there is no BCNF violation.
   c. N/A
   d. N/A

In conclusion there is no BCNF violation in the relations and functional dependencies. This happens because when working on the E/R diagram it is easy to see the pit falls and even automatically apply BCNF normalization. For instance, Class and Course were separated after drawing the E/R diagram with them being one entity set initially, it was not all that apparent at the beginning but before starting the BCNF section it was obvious that having one entity set creates redundancy, so the single entity set was divided to two entity sets, Class and Course.

Any BCNF set is in Third Normal Form (3NF) therefore there is no need to normalize in 3NF. The only benefit of 3NF is preserving the functional dependencies, however no functional dependency was removed for these relations, so there would be no difference to have them in 3NF form in this case. BCNF is stricter and better for most application from practical point of view.

As for Fourth Normal Form (4NF), 4NF is stricter than BCNF, however for this case the current BCNF is already in 4NF form as well because there is no multivalued dependency (MVD). 4NF optimizes the MVDs which BCNF does not, however in this case there is no MVD, so 4NF of this case would be same as its BCNF.

4. SQL on MySQL Local Server

a. Setting up a MySQL Database

Starting with zero background with a database, setting a database initially seemed like a very hard task, but MySQL Workbench makes the process very easy, especially because it runs on Windows and has rich graphical user interface. I noticed that MySQL Workbench allows us to draw diagram, I tried to draw the diagram for my database but the notation for one-to-many and many-to-many were different and since I already had the E/R diagram, I decided to stick to my own version which uses similar notations as what we discussed in class. Since I did not used the E/R model in MySQL GUI, I decided to create my tables in MySQL Workbench.

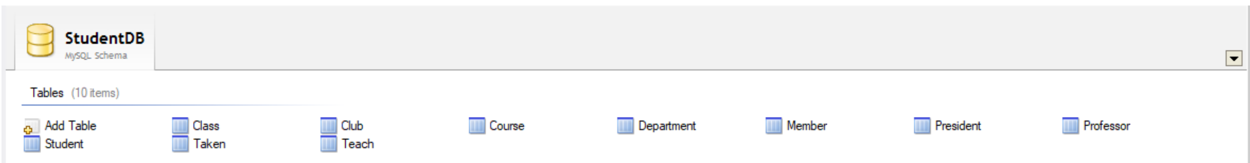


Figure 1 MySQL Workbench View of Database Tables.

While the view above shows all the tables after they have been constructed in the GUI, the following figures show how each table was setup. Yellow icon represents Primary Key, red icon represents foreign keys, and blue icon represents other attributes. When we set the foreign keys, they must have unique name, to achieve this I have added "fk_" as prefix and table name as postfix. If a table has more keys, that can be indicated by using Unique ("UQ"), as it can be seen in Figure 7, where the values of email and club names must be unique at any given time as discussed earlier.

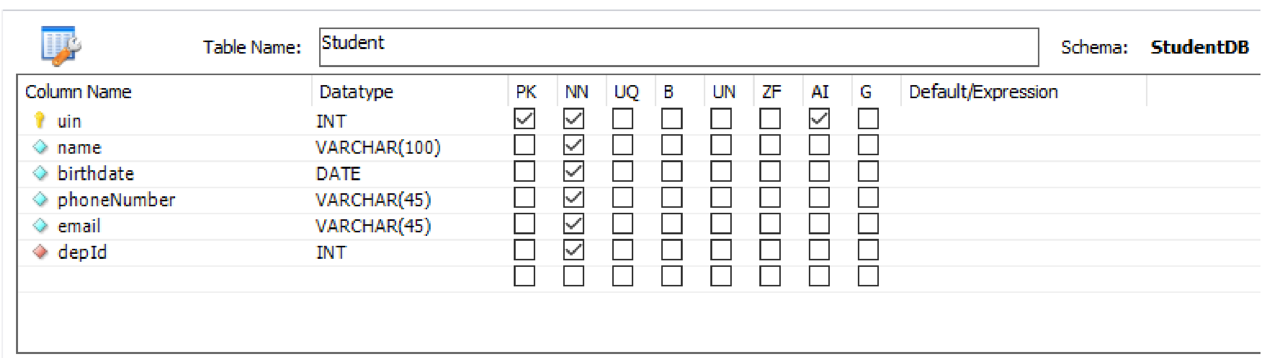


Figure 2 Student(uin, name, birthdate, phoneNumber, email, depId) table in MySQL Workbench.

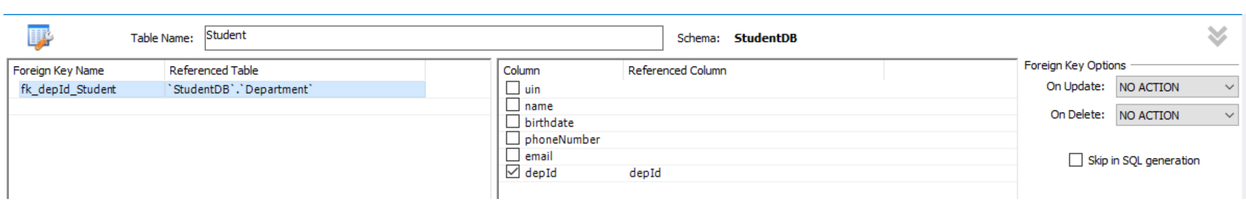


Figure 3 setting "depId" as a foreign key for "Student" table in MySQL Workbench.

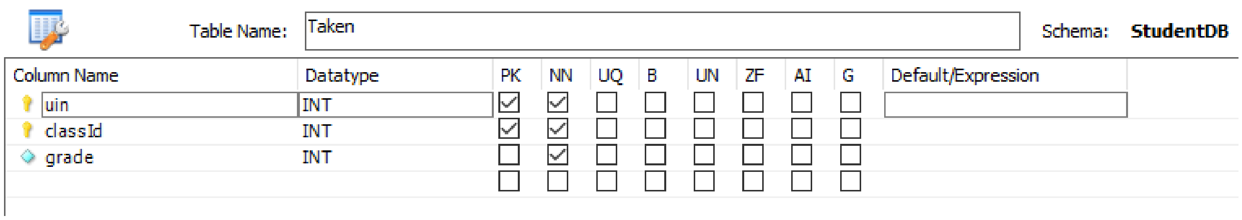


Figure 4 Taken(uin, classId, grade) table in MySQL Workbench.

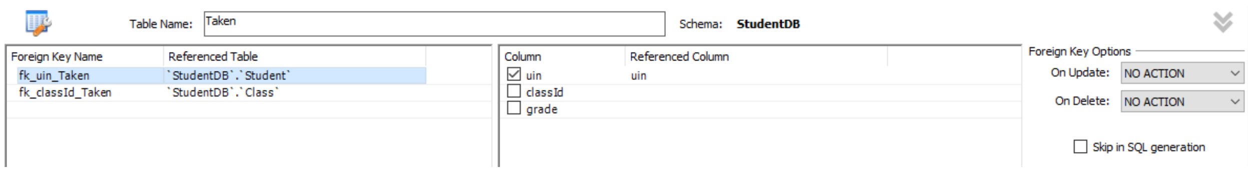


Figure 5 setting "uin" as a foreign key for "Taken" table in MySQL Workbench.

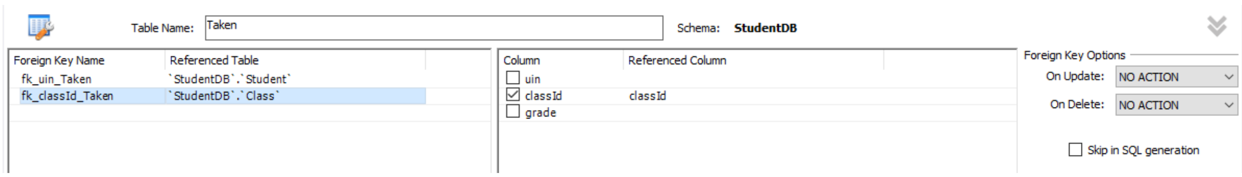


Figure 6 setting "classId" as a foreign key for "Taken" table in MySQL Workbench.

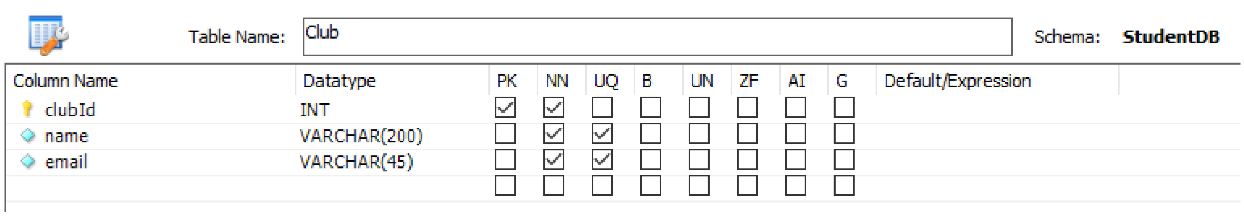


Figure 7 Since each table can have one Primary Key, the rest of the keys must be selected as Uneque ("UQ"). This allows the database to reject non-unuque values for these attributes .

For sake of simplicity all human (i.e. student and professor names) names are set to have maximum length of 100, while other names are set to have 200- character limit (i.e. course's, department's and club's name). Phone numbers and emails are all set to have 45-character limit. There is only one primary key that is not integer, which is "courseCode" and 15-character limit is set for "courseCode".

While MySQL Workbench creates everything from the graphical interface, I also generated the SQL script and checked to make sure it is correctly generated and learn if there are features that I am not aware of. The SQL file is submitted with other files, the name and path for the file is "MySQL\SQL Script to define the SQL server.sql", and the C# program that will be discussed later, has the capability to generate the database.

b. Data Generation

In order to easily test the database and make sense out of data, I created an excel file called "StudentDatabaseSample.xlsx", located at "project1\GUI\Student Management System\bin\Debug". I collected the first names and family names from [1]. The departments are based on Brockport State University Of New York, their department names are available at [2].

The course names are extracted with some preprocessing from “Download all 2020-21 courses as CSV” from Wisconsin Department of Public Instruction. (the file [3], the website [4]). Club names are extracted from Brandies University’s website [5]. For emails, I have developed a code that based on the name of the individual, department or club generates an email. For phone numbers they are randomly generated. Course codes have several 3-letter prefixes followed by a hyphen and 4 digits. These 3-letter prefixes are “ABC”, “DEF”, “GHI”, “JKL”, “MNO”, “PQR”, and “STU”. An example of a course code would be “ABC-4186”. Students grades are randomly generated but they are designed to have an average around 82.5% without a uniform distribution. Students’ and professors’ birthdays are selected to make sense with professor usually being older. The semester and year are randomly selected, for semester between 1 to 3 and for year from 2010 to 2019. With probability of almost 3 percent some classes have more than one instructor. Finally, to have maximum testing capability, the data generation setting can be set to generate different number of tuples. The figure below shows the GUI’s data generation option with default numbers. The default numbers generate 15 department, 150 professors, 1500 students, 60 courses (different from classes), 300 classes (specific classes that student take), 50 clubs, 4500 tuples in “taken” table, 400 tuples in “members”, 50 tuples in “president”, at least 300 tuples in “teach” (since courses randomly have more than on instructor it would vary from run to run). These number can be altered to achieve different data and most importantly they always make sense.

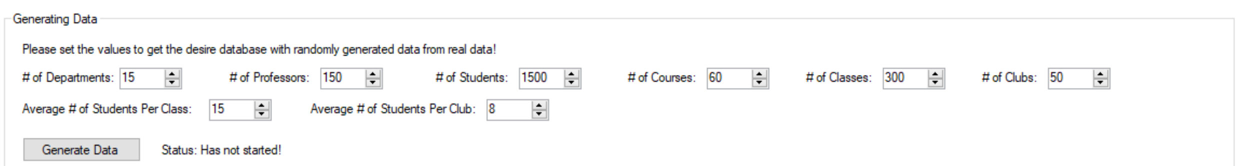


Figure 8 The part of the GUI that generates data, it is currently shown with default value, but they can be altered

The data generated could be easily verified by MySQL Workbench, before developing my application to read the database. Also, I have set my C# program to capture all exception and as a result, all unique value Few are guaranteed to be unique and if a random part is duplicated, the random runs again to make sure that does not happen. The data generator also makes sure all foreign keys are set correctly and for most queries there will be some results. The random generated parts are random but withing values that produce meaningful result. The figures below are added for demonstration of output data from MySQL Workbench, in these figures “depId” can be seen as the primary key of the department table and foreign key for “course” and “student” tbale.

depId	name	phoneNumber	email
100	Academic Affairs	(659)283-3898	AcademicAffairs@uni.org
101	Academic Success Center	(114)049-3432	AcademicSuccessCenter@uni.org
102	Accountability and Assessment	(977)095-7586	AccountabilityandAssessment@uni.org
103	Accounting, Economics & Finance	(991)176-9557	Accounting,Economics&Finance@uni.org
104	Administration and Finance	(640)157-4464	AdministrationandFinance@uni.org
105	Admissions Undergraduate	(511)048-7823	AdmissionsUndergraduate@uni.org
106	Advancement	(580)380-0338	Advancement@uni.org
107	Affirmative Action	(988)664-6652	AffirmativeAction@uni.org
108	African & African-Amer Studies	(138)317-8169	African&African-AmerStudies@uni.org
109	Anthropology	(813)217-3624	Anthropology@uni.org
110	Art	(711)846-3549	Art@uni.org
111	Arthur O Eve - E O P Program	(213)031-9621	ArthurOEve-EOPProgram@uni.org
112	ASC Advisement and Retention	(396)398-4648	ASCAdvisementandRetention@uni.org
113	ASC Student Accessibility Svcs (...	(203)158-2094	ASCStudentAccessibilitySvcs(Officef@uni.org
114	ASC Tutoring	(427)677-7289	ASCTutoring@uni.org
NULL	NULL	NULL	NULL

Figure 9 The table "department" after generating data

courseCode	name	credit	depId
ABC-4186	Visual Arts-Drawing/Painting	2	100
ABC-6524	AP U.S. History	5	112
ABC-6793	AP Human Geography	1	100
ABC-9218	Arabic Literature	2	107
DEF-1634	Particular Topics in Restaurant, Food and Beverage Services	1	113
DEF-2834	IB Language A: Literature--Italian	4	112
DEF-4537	Particular Topics in Restaurant, Food and Beverage Services	5	100
DEF-4923	Integrated Science	5	106
DEF-9107	Business Math with Algebra	3	114
DEF-9328	Marketing-Merchandising	4	101
GHI-1028	Restaurant, Food and Beverage Services-Independent Study	3	109
GHI-2464	IB Language A: Language and Literature--Filipino	4	113
GHI-2763	Emerging Technologies	1	106
GHI-3092	Music-Other	5	108
GHI-3336	Calculus	5	101
GHI-4675	Music (kindergarten)	5	102
GHI-4930	Individual Technique-Vocal Music	1	110
GHI-6308	Mathematics (grade 3)	2	100
GHI-9583	Particular Topics in Cosmetology	5	114
GHI-9729	Grade 6	3	105
JKL-1200	Scriptures	5	108
JKL-1676	Macroeconomics	4	111
JKL-3756	Classical Greek II	2	112
JKL-4231	Restaurant Management and Operations	2	108
JKL-6148	IB Primary Years Program	2	112

course 1

Figure 10 The table "course" after generating data

Result Grid | Filter Rows: | Edit: | Export/Import:

uin	name	birthdate	phoneNumber	email	depId
10000	Carma Marvel	1998-05-08	(249)922-8583	CarmaMarvel@uni.org	107
10001	Donette Nu	1996-01-22	(625)691-6039	DonetteNu@uni.org	110
10002	Willodean Paris	1998-08-18	(561)803-9466	WillodeanParis@uni.org	107
10003	Ozell Judy	1996-09-22	(485)216-2078	OzellJudy@uni.org	109
10004	Carin Charlene	1994-10-05	(785)231-7320	CarinCharlene@uni.org	100
10005	Teri Kristeen	1994-06-24	(390)813-4731	TeriKristeen@uni.org	102
10006	Stephaine Dulce	1997-12-08	(836)794-4212	StephaineDulce@uni.org	100
10007	Shenika Corinne	2000-06-10	(480)539-0347	ShenikaCorinne@uni.org	113
10008	Carma Janey	1996-08-18	(275)277-8858	CarmaJaney@uni.org	111
10009	Cristal Lezlie	1993-02-14	(777)853-9832	CristalLezlie@uni.org	104
10010	Talia Dalene	1992-04-17	(639)932-5692	TaliaDalene@uni.org	100
10011	Ressie Timothy	1991-01-04	(644)177-0776	RessieTimothy@uni.org	113
10012	Raymon Fausto	1997-04-08	(613)931-1421	RaymonFausto@uni.org	113
10013	Irma Carissa	1995-09-08	(226)170-8832	IrmaCarissa@uni.org	102
10014	Lashawnda No...	1990-04-11	(978)971-7420	LashawndaNovella@uni.org	114

Figure 11 The table Student after generating data

5. My .NET C++/C# Windows Application

a. Software and external libraries used

I have used Visual Studio 2019 and C# Windows Application, to assist me with the graphical side of the application. I have used Connector/NET 8.0.22 from [6] to connect my C# application to locally hosted MySQL server. I have used “microsoft.office.interop.excel” to parse the excel file [6]. Everything else I have used are parts of built in .NET libraries.

b. Connecting to Database, generating and deleting data

I started by focusing on connecting to the data base and trying to read and write data to it. Also, I made it flexible so that I can test different things, like accessing the database from different account and generating different data sets with different sizes. Figure 12 shows the first page of the application before connecting to the server. Since we are not connected to a server all options are turned off to avoid confusion and unwanted errors. The default values are used to connect directly to my local database. Figure 13 shows the application after connecting to the server. Take note that again default values are loaded for quick testing, but they can be changed easily. They have boundary checking considering that the data in the excel file is limited. Although it does a great deal of trying to extrapolate from the excel file. Once you deleted or generate data the status text, gives you information about the process. For deleting it tells you each table that has been deleted fully and for data generation, it shows every tuple that is being created and sent to server. Also take not that the database used for this project can be created by clicking on “Create Database”.

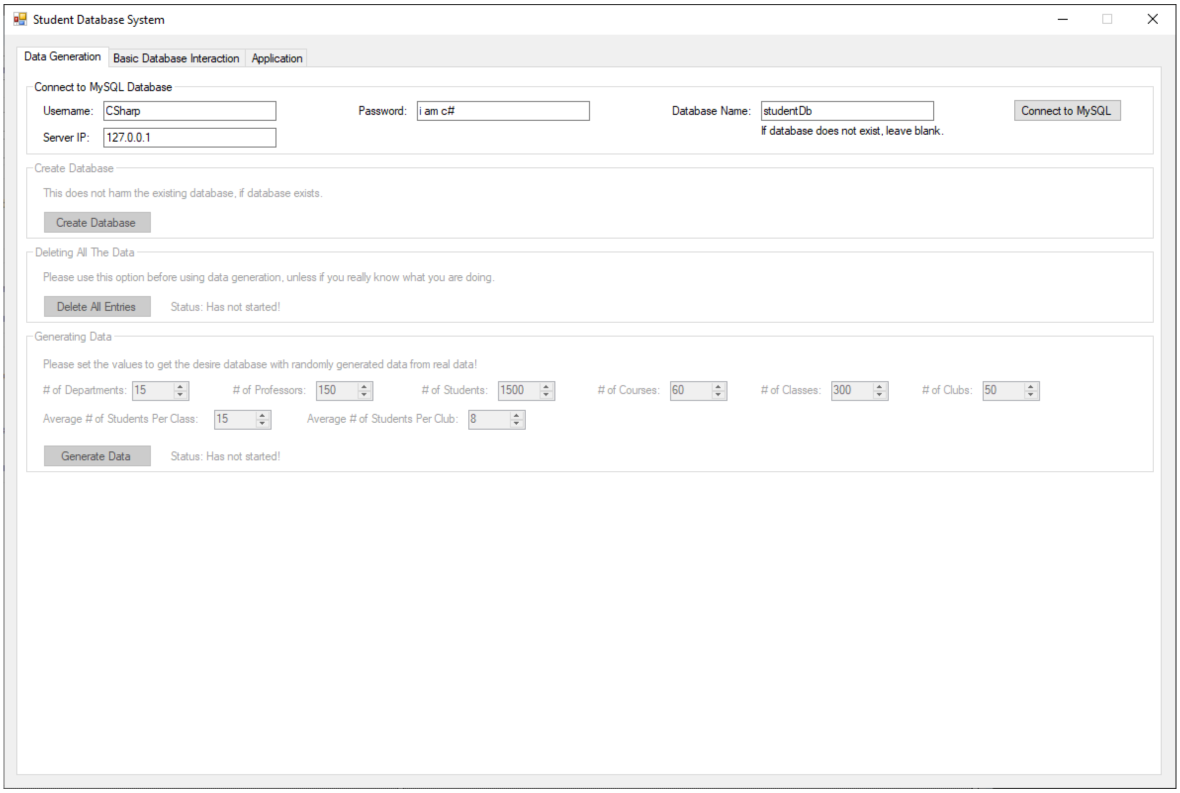


Figure 12 The first page of the application before connecting to server.

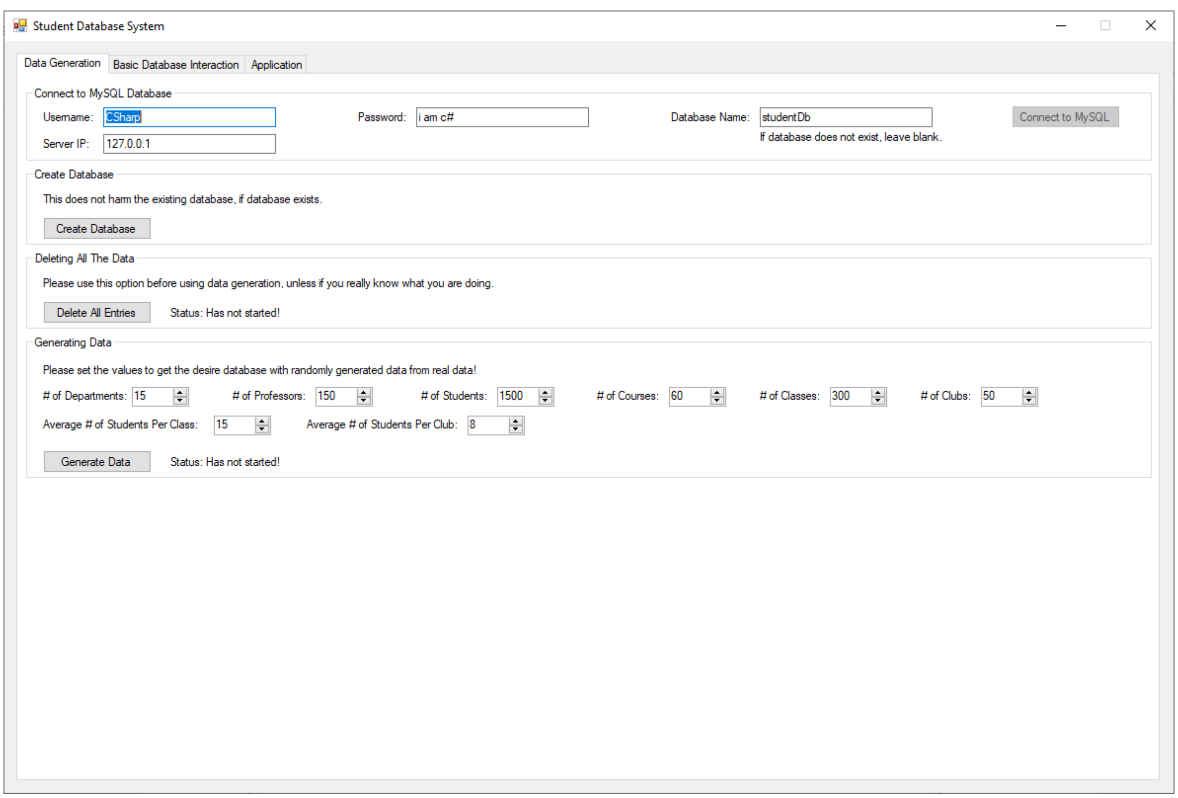


Figure 13 The first page of the application after connecting to the server

c. Easy editing and easy look up for foreign keys

After getting the previous part done, I made an interface to easily add, edit, or delete tuples either as a single tuple or as a table. But then soon I noticed that not having foreign key the process become very difficult. So, I added table to look up foreign keys on the same page, where the main table is being edited. Take note that the user can use the entries provided on top to add, edit or delete tuples, or they can directly change the information on the table and click on "Update Table". By clicking on "Refresh Table" the table get read back from database. I track all changes that are made to the table as you use the tuple features to invalidate the table. Once a table is invalidated, the "Update Table" button become disabled so that you cannot write back to the database, before "Refreshing Table", The look up table(s) on the right can be used to look up foreign keys. They do not stay active, so changes from other tabs does not update their values. This is because of the resource constrains. It would be resource consuming to track all the look up tables at all time. In this section I have tried to make sure the interface remains similar to the database itself. That is why the attributes maintain their original names and foreign keys are not joined because that would change the actual presentation of the tables in the data base. To help the users Primary Key is show in Red while foreign keys are shown in blue.

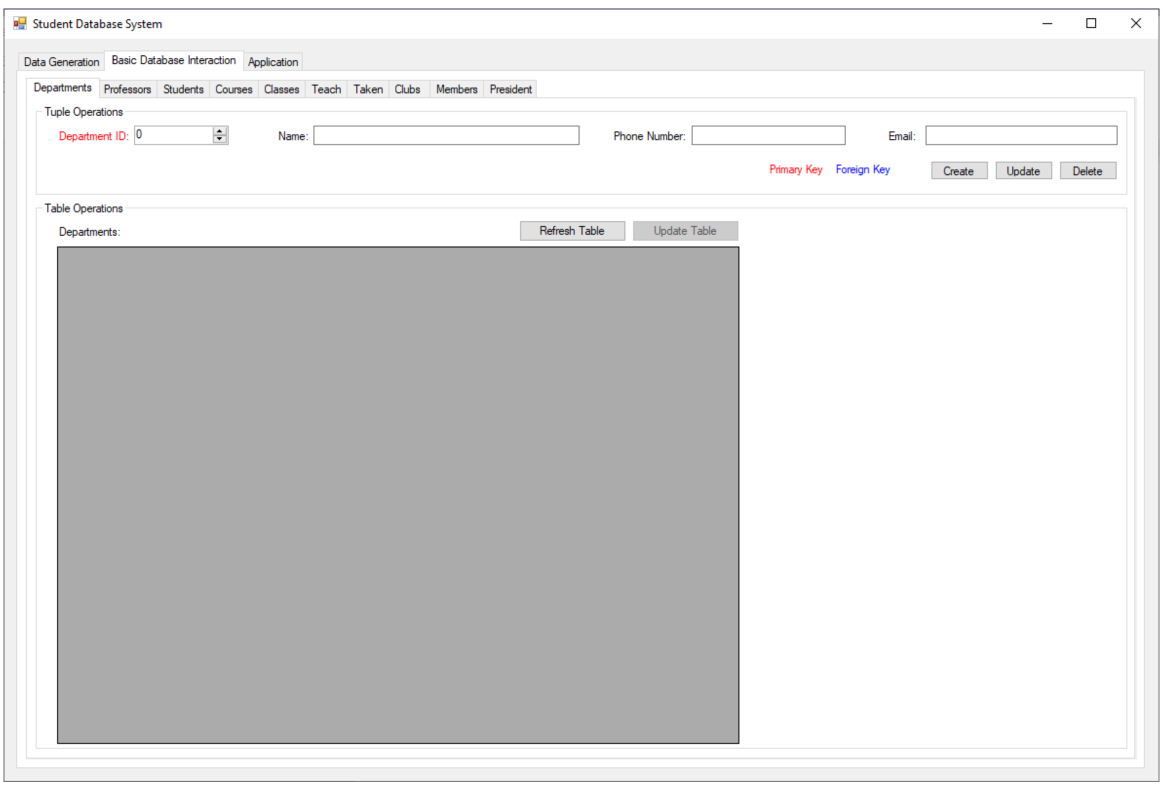
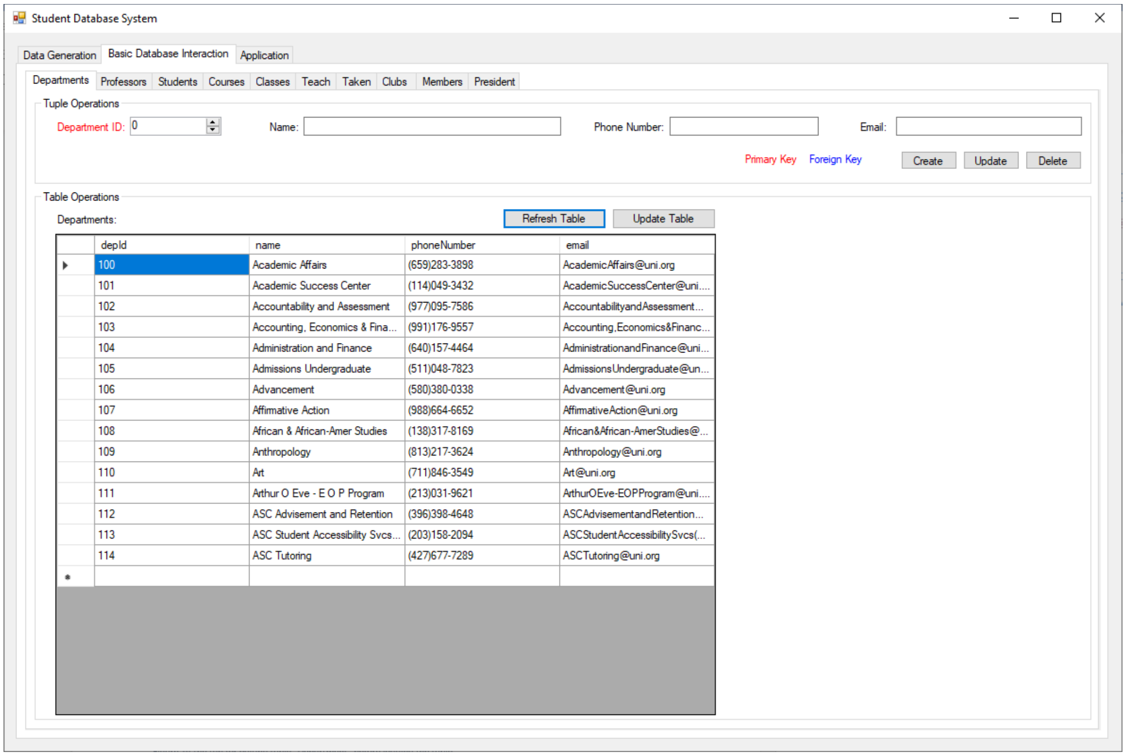
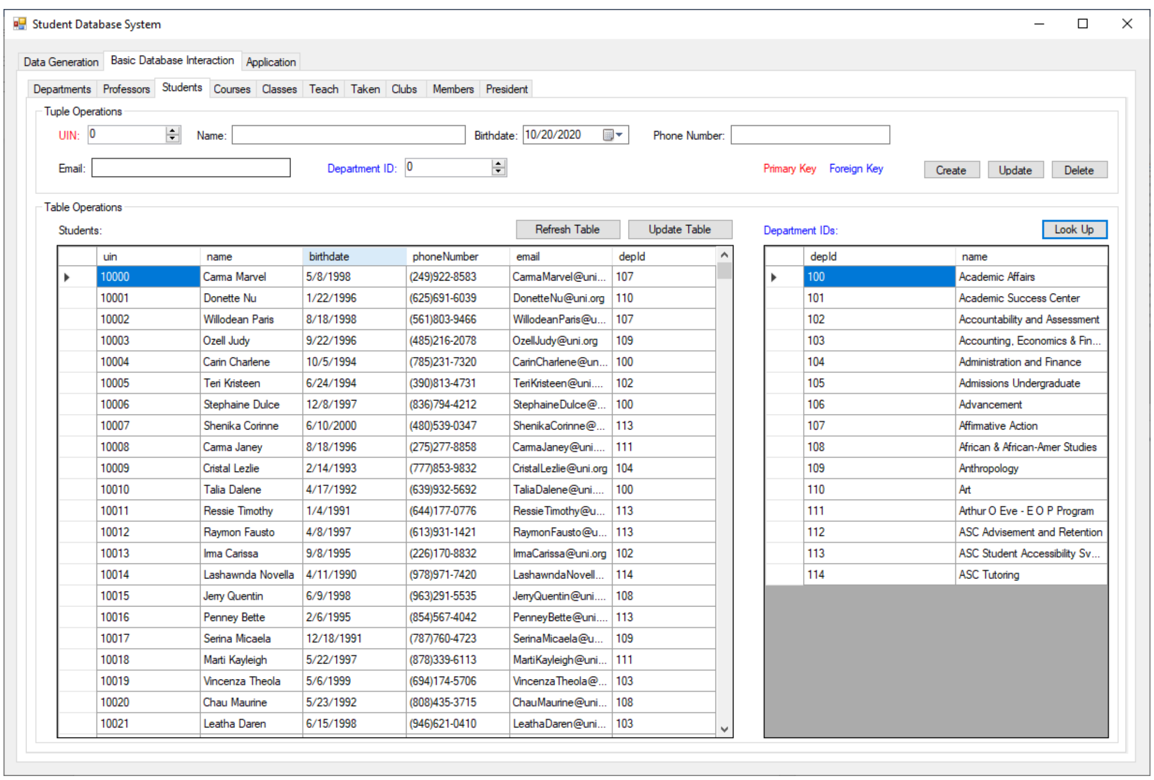


Figure 14 the tab for editing table "Department" before loading the table



depId	name	phoneNumber	email
100	Academic Affairs	(659)283-3898	AcademicAffairs@uni.org
101	Academic Success Center	(114)049-3432	AcademicSuccessCenter@uni....
102	Accountability and Assessment	(977)095-7586	AccountabilityandAssessment...
103	Accounting, Economics & Fina...	(991)176-9557	Accounting,Economics&Financ...
104	Administration and Finance	(640)157-4464	AdministrationandFinance@uni...
105	Admissions Undergraduate	(511)048-7823	AdmissionsUndergraduate@un...
106	Advancement	(580)380-0338	Advancement@uni.org
107	Affirmative Action	(988)664-6652	AffirmativeAction@uni.org
108	African & African-Amer Studies	(138)317-8169	African&African-AmerStudies@...
109	Anthropology	(813)217-3624	Anthropology@uni.org
110	Art	(711)846-3549	Art@uni.org
111	Arthur O Eve - E O P Program	(213)031-9621	ArthurOEve-EOPProgram@uni....
112	ASC Advisement and Retention	(396)398-4648	ASCAdvisementandRetention...
113	ASC Student Accessibility Svcs...	(203)158-2094	ASCStudentAccessibilitySvcs(...
114	ASC Tutoring	(427)677-7289	ASCTutoring@uni.org

Figure 15 the tab for editing table "Department" after loading the table



uin	name	birthdate	phoneNumber	email	depId
10000	Carma Marvel	5/8/1998	(249)922-8583	CarmaMarvel@uni...	107
10001	Donette Nu	1/22/1996	(625)691-6039	DonetteNu@uni.org	110
10002	Willodean Paris	8/18/1998	(561)803-9466	WillodeanParis@u...	107
10003	Ozell Judy	9/22/1996	(485)216-2078	OzellJudy@uni.org	109
10004	Carin Charlene	10/5/1994	(785)231-7320	CarinCharlene@un...	100
10005	Teri Kristeen	6/24/1994	(390)813-4731	TeriKristeen@uni....	102
10006	Stephaine Dulce	12/8/1997	(836)794-4212	StephaineDulce@...	100
10007	Shenika Corinne	6/10/2000	(480)539-0347	ShenikaCorinne@...	113
10008	Carma Janey	8/18/1996	(275)277-8858	CarmaJaney@uni....	111
10009	Cristal Lezlie	2/14/1993	(777)853-9832	CristalLezlie@uni.org	104
10010	Talia Dalene	4/17/1992	(639)932-5692	TaliaDalene@uni....	100
10011	Ressie Timothy	1/4/1991	(644)177-0776	RessieTimothy@u...	113
10012	Raymon Fausto	4/8/1997	(613)931-1421	RaymonFausto@u...	113
10013	Irma Carissa	9/8/1995	(226)170-8832	IrmaCarissa@uni.org	102
10014	Lashawnda Novella	4/11/1990	(978)971-7420	LashawndaNovell...	114
10015	Jerry Quentin	6/9/1998	(963)291-5535	JerryQuentin@uni....	108
10016	Penney Bette	2/6/1995	(854)567-4042	PenneyBette@uni....	113
10017	Serina Micaela	12/18/1991	(787)760-4723	SerinaMicaela@u...	109
10018	Marti Kayleigh	5/22/1997	(878)339-6113	MartiKayleigh@uni...	111
10019	Vincenza Theola	5/6/1999	(694)174-5706	VincenzaTheola@...	103
10020	Chau Maurine	5/23/1992	(808)435-3715	ChauMaurine@uni...	108
10021	Leatha Daren	6/15/1998	(946)621-0410	LeathaDaren@uni...	103

depId	name
100	Academic Affairs
101	Academic Success Center
102	Accountability and Assessment
103	Accounting, Economics & Fin...
104	Administration and Finance
105	Admissions Undergraduate
106	Advancement
107	Affirmative Action
108	African & African-Amer Studies
109	Anthropology
110	Art
111	Arthur O Eve - E O P Program
112	ASC Advisement and Retention
113	ASC Student Accessibility Sv...
114	ASC Tutoring

Figure 16 The table for "student" one the left being edited while "department" table's key information is available on the right so that the user can match the foreign keys

d. Running complex queries

In this section I made more advanced quires for student, professor and courses. Take note that each table in GUI is a different query, but each table is a single query.

i. For student (refer to Figure 21):

1. Personal Information: The student table does not show the students department name. So, in this query, I read the department name from "department" table and add the name there.
2. GPA and Credit Hours: I calculate the student's GPA, which is weighted since courses have different credit hours. The total number of credits is also calculated and shown in the table. Lastly all credits with score above 60% are considered a pass, and they are counted separately. Take note that this is one nested query and no data manipulation is done. (Figure 17)
3. Courses Taken: In here you can see all the classes and courses taken by the student, the time that he/she has taken the course, his/her grade, the class's average, the top score in class and the professor teaching the course. Again, all of this is done in a single query without data manipulation at C#'s end. (Figure 18)
4. Club Membership: In this query we find all the clubs that the student is part of.
5. Club President: In this query we find the Clubs that the student is the president for.

ii. For Professor (refer to Figure 22)

1. Personal Information: Given the Professor ID, the application loads the professor's personal information. This is pretty easy since all data is in one table.
2. Courses Taught: Shows the courses the professor has taught with all the information about the class average and also the top score. The query for this is different from the one used for student. I have used "group by" since I could group by classID. This would not be possible for student, because in student's case, we are interested in one student's record. (Figure 19)

iii. For Courses (refer to Figure 23)

1. Course History and Data: In this query, I show the history of subject, being taught by different professors, the class average and top score in those classes. Again, this is a single query. (Figure 20)

```sql
SELECT student.name as 'Name', (sum(taken.grade * course.credit)/(sum(course.credit))) as 'GPA',
sum(case when taken.grade > 60 then course.credit end) as 'Total Credit Passed', sum(course.credit) as 'Total Credit Taken'
FROM student, taken, course, class
WHERE course.courseCode = class.courseCode AND class.classId = taken.classId AND taken.uin = student.uin and student.uin = uin;
```

Figure 17 Query to calculate student GPA and number of credits (passed credits and total)

```sql
SELECT class.classId as 'Class ID', course.courseCode as 'Course Code', course.name as 'Course Name', class.year as 'Year',
class.Semester as 'Semester', course.credit as 'Credit Hours', taken.grade as 'Grade', allAverages.courseAverage as 'Class Average',
allAverages.bestGrade as 'Top Score', professor.name as 'Professor Name'
FROM professor, taken, class, course, teach, (SELECT classId, (sum(taken.grade)/count(taken.uin)) as courseAverage,
max(taken.grade) as bestGrade FROM taken group by classId) as allAverages
WHERE  allAverages.classId = class.classId AND teach.profId = professor.profId AND teach.classId = class.classId AND
course.courseCode = class.courseCode AND class.classId = taken.classId AND taken.uin = uin;
```

Figure 18 Query for courses taken by a student

```sql
SELECT class.classId as 'Class ID', course.courseCode as 'Course Code', course.name as 'Name', class.year as 'Year', class.semester  as 'Semester',
course.credit as 'Credit Hours', (sum(taken.grade)/count(taken.uin)) as 'Class Average', max(taken.grade) as 'Top Scpre'
FROM professor, teach, class, course, taken
WHERE taken.classId = class.classId  AND class.courseCode = course.courseCode AND class.classId = teach.classId AND teach.profId = professor.profId AND professor.profId = profId
group by class.classId;
```

Figure 19Query for Courses taught by the given professor

```sql
SELECT class.classId as 'Class ID', department.name as 'Department', course.courseCode as 'Course Code', course.name as 'Course Name', course.credit as 'Credit Hours',
class.year as 'Year', class.Semester as 'Semester', allAverages.courseAverage as 'Class Average', allAverages.bestGrade as 'Top Score', professor.name as 'Professor Name',
professor.email as 'Professor Email', professor.profId as 'Professor ID'
FROM department, professor, course, class, teach,
(SELECT classId, (sum(taken.grade)/count(taken.uin)) as courseAverage, max(taken.grade) as bestGrade FROM taken group by classId) as allAverages
WHERE department.depId = course.depId AND allAverages.classId = class.classId AND professor.profId = teach.profId AND
teach.classId = class.classId AND class.courseCode = course.courseCode AND course.courseCode = courseCode;
```

Figure 20 Query for course history

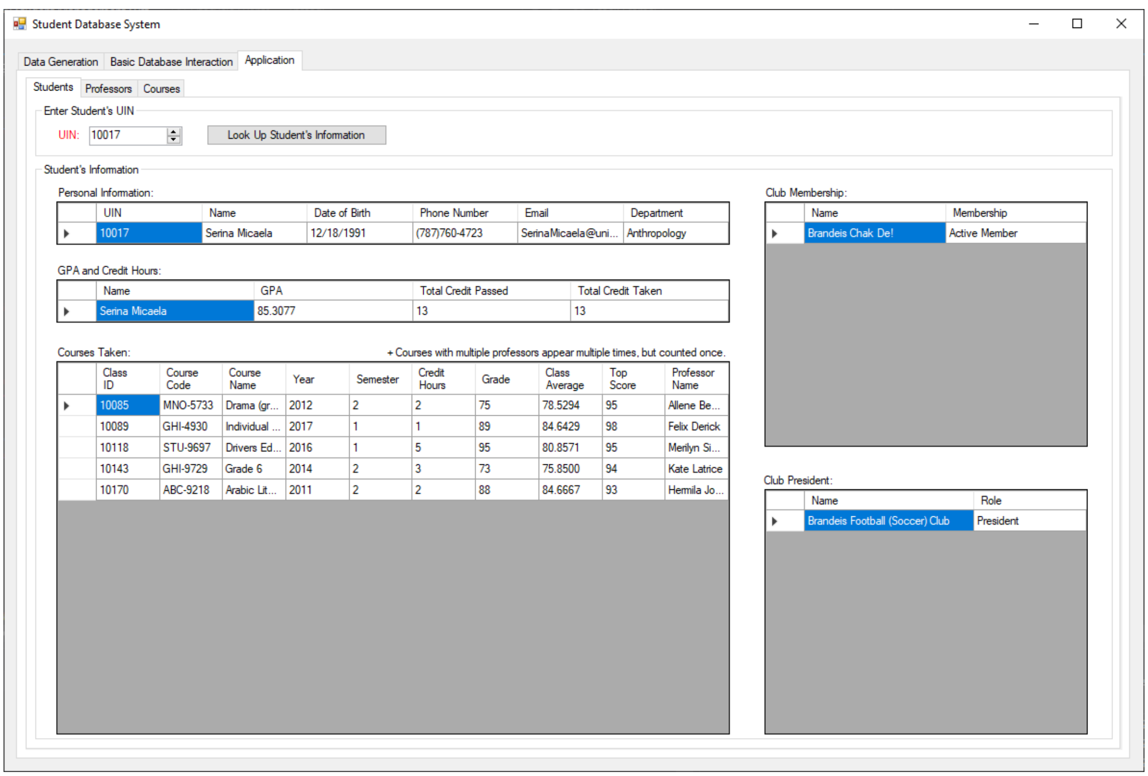


Figure 21 Advanced queries ran on student to find various information about his grades and course he/she has taken

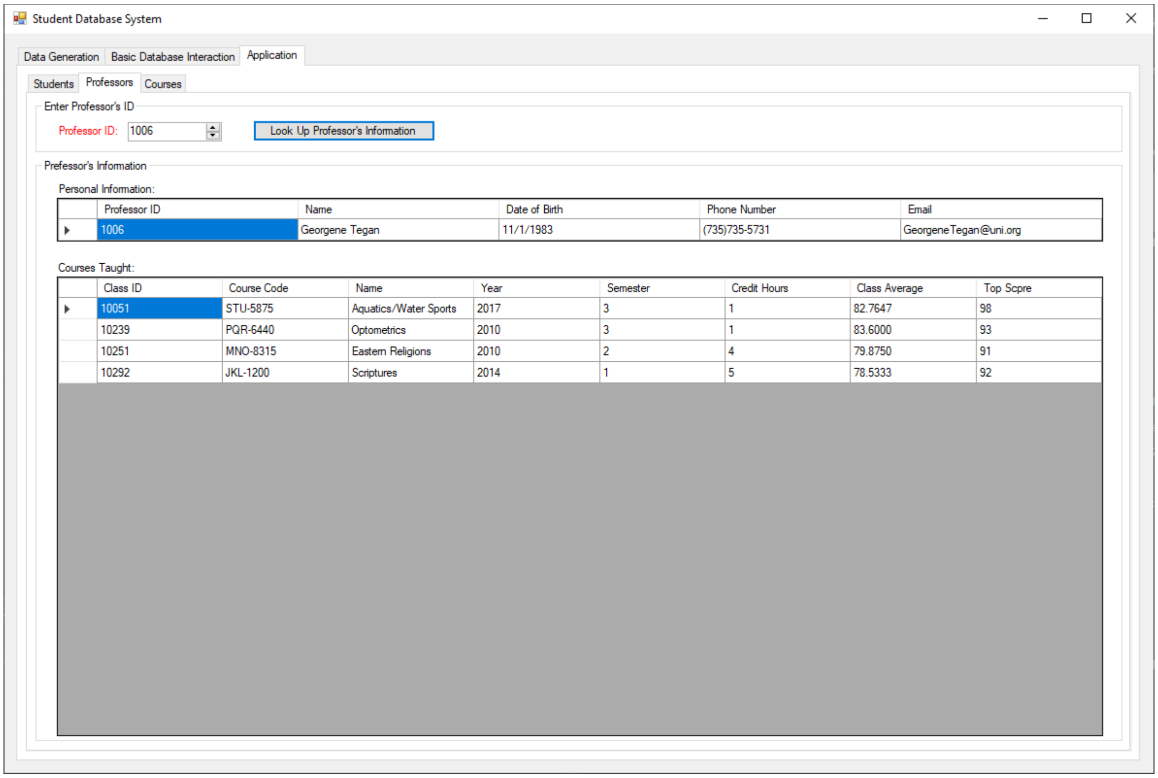


Figure 22 The professor's teaching history and students performance in the classes the professor has taught

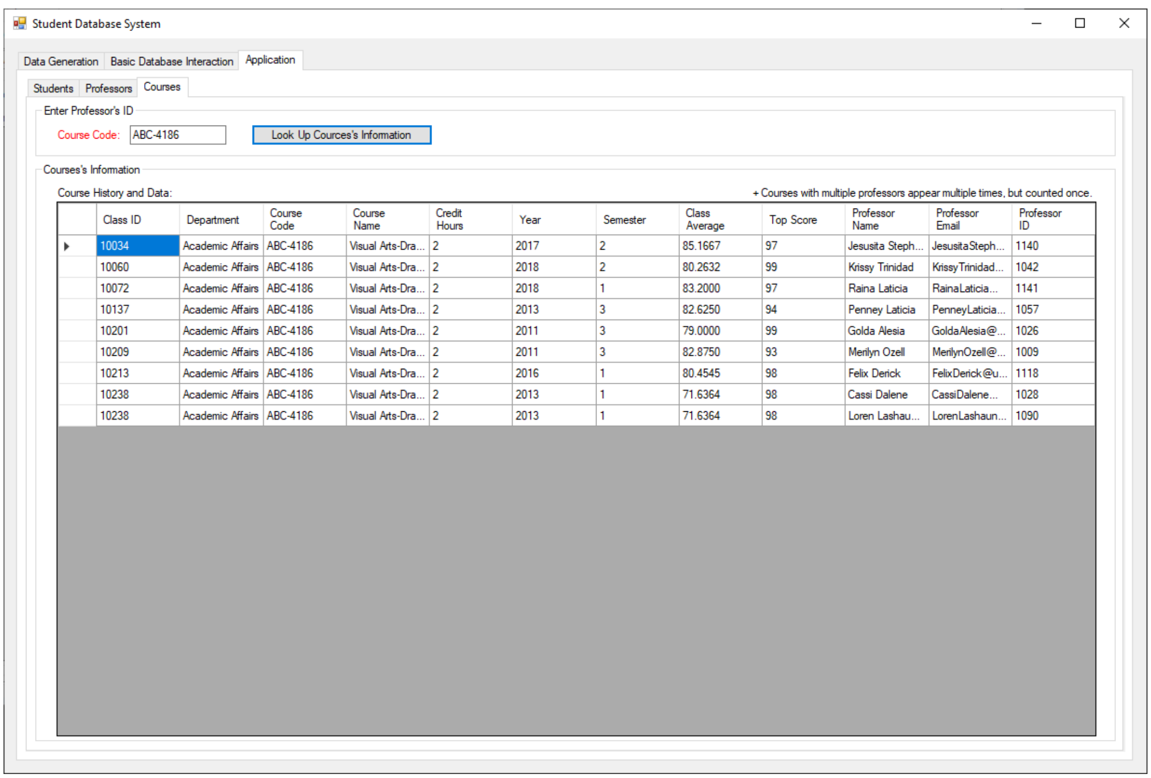


Figure 23 The course history page showing all the times that a course was offered and students' performance and the instructors' information

[1] https://www.briandunning.com/sample-data/

[2] https://www.brockport.edu/support/human_resources/forms/department_names.html

[3] https://dpi.wi.gov/sites/default/files/ed-fi/courses.csv

[4] https://dpi.wi.gov/wise/data-elements/coursereference

[5] https://www.brandeis.edu/clubs/

[6] https://dev.mysql.com/downloads/connector/net/

[7] https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/interop/how-to-access-office-onterop-objects