Names: Cameron Ellis, Sneha Reddy, Micah Remus

Group 16

CS 3380

Group 16 Project Phase 3 – School District Database

PROBLEM STATEMENT:

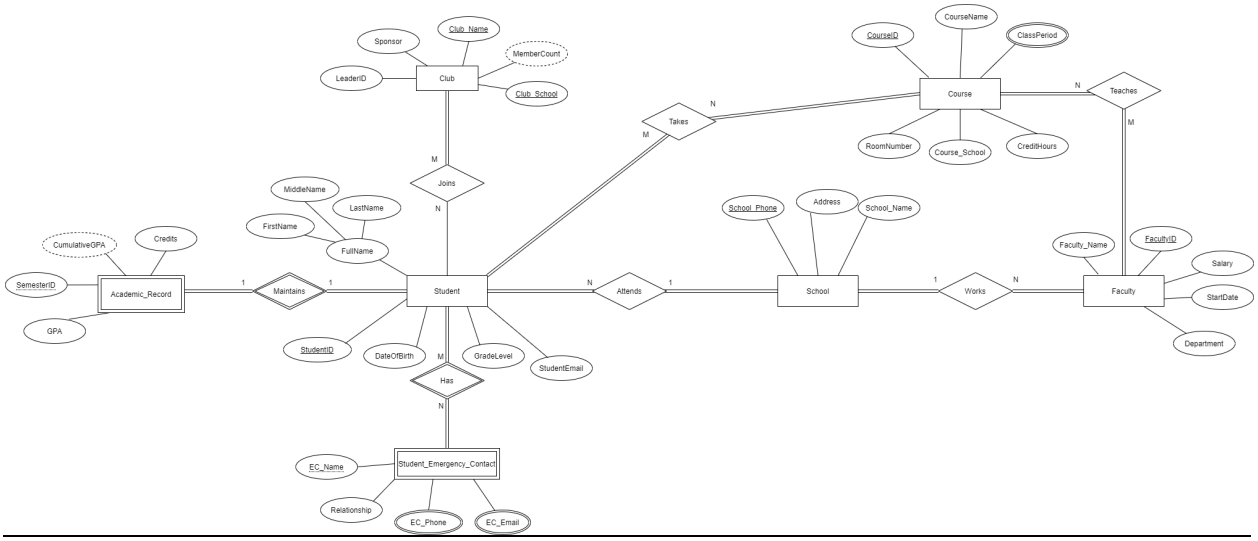
The database we have designed is purposed to contain information for a school district in order to keep track of all information related to its schools, students, faculty members, courses, and clubs. The information in the database that is kept for the students is their personal information including their name, email, student ID, date of birth, and grade level. The database also keeps track of each student's academic record, their emergency contacts, clubs they've joined, which school they are attending, and the courses they are taking/have taken. For the faculty members, each record has their faculty ID, name, salary, start date, and the department they work in. The database also keeps track of which school each faculty member works for in the district. Lastly, for the courses we keep track of which students are taking them, who is teaching them, and basic information including the course ID, name, period(s) it is offered, room number, how many credit hours it is worth, and which school it is offered at. This database system is essential because without it, it would be next to impossible for there to be a centralized copy of school information across the district that is able to be accessed at any given time. If the district relied upon hard copies of all of this information, not only would it be detrimental if it was lost, but it would be very difficult to access the information easily without having to dig through piles of papers and physically travel to each school to get the most up to date information. It allows students, faculty members, and central chairmen of a school district to have back up files for essential information that is relevant and constantly updated which can be pulled from and used at a moment's notice.

CONCEPTUAL DATABASE DESIGN:

EER Diagram Description:

In this database, the information for an entire School District is to be stored. In this diagram, each Student has their Name (First, Middle, Last), StudentID, Date of Birth, Grade Level, and Student Email kept track of. The database also keeps track of the Student's Academic Record, Clubs they've joined, Courses they are taking, the School in the district they attend, and any Emergency Contact Information. For the Academic Record of the student, the SemesterID in combination with the Student's Student ID uniquely identifies each record. It also includes the Semester GPAs of the student, the Cumulative GPA, and how many Credits the student has taken in a semester. For the Clubs, what is kept track of is its Name, the ID number of the student Leader of the Club, the Sponsor (could be teacher but doesn't have to be, could be a parent or coach), the Member Count, and the School for which the club is assigned to. For the Student Emergency Contact, their Name, Relationship to the student, Phone Number (which can have multiple values for work, cell, etc.), and Email are kept track of (which can also have multiple values for personal email, work email, etc.). For the Courses, what is kept track of is the Course Name, CourseID, the Room Number for the room it is taken in, how many Credit Hours it is worth, the Class Period(s) it is offered, and the School (name) it is offered at. For the Schools, what is kept track of is its Phone Number, Address, and Name. Lastly, for the Faculty, what is kept track of is their Name, FacultyID, Salary, the day they started, which classes they teach, and the department of their school they are in.

EER Diagram:



Assumptions:

- Student age can be derived from their birth date
- Every student must take courses, and each course must be taken by students
- Students take multiple classes and classes are taken by multiple students
- Each class can be taught multiple different class periods
- Not all the same classes must be taught at every school
- Faculty must teach classes, and classes must be taught by faculty
- Faculty members can teach multiple classes, and each class can be taught by multiple teachers
- There are multiple departments per one school
- Every student must go to a school, and every school must have students go to it
- Faculty members must work at a school and schools must have faculty members work at them
- Faculty members only work at one school

- Every club must have students in it, but not every student has to join a club
- There are many students for one club, and each student can join multiple clubs
- Member count of a club can be derived from how many students are in the club
- There is one academic record for one student
- Every student must have an academic record, and every academic record must have a student attached to it
- Cumulative GPA is derived from Semester GPAs
- Academic record is a weak entity type that uses SemesterID as a partial key and StudentID from the Student entity to uniquely identify it.
- Each student must have an emergency contact, and every emergency contact must be linked to a student
- Each student can have multiple emergency contacts, and each emergency contact can be related to multiple students (in the case of siblings going to the same school)
- Emergency contact is a weak entity type and is uniquely identified by the minor key of the name and the StudentID for the student they are connected to

APPLICATION PROGRAM DESIGN:

Function Descriptions:

insert Function:

- **Description:** This function is a general insert function used to insert a new tuple into any of the relations in the DB (Student, Course, School, etc.)

- **Generalized Code:**

```
INSERT INTO TABLE_NAME
(Attr_1, Attr_2, Attr_3, ... , Attr_n)
VALUES
(T1A_1, T1A_2, T1A_3, ..., T1A_n),
(T2A_1, T2A_2, T2A_3, ..., T2A_n),
(T3A_1, T3A_2, T3A_3, ..., T3A_n),
...
(TnA_1, TnA_2, TnA_3, ..., TnA_n);
```

- **Input:** All attributes for the particular relation the user is wanting to insert a new tuple into.
- **Steps:**
    - The user selects which relation they want to enter a new tuple into.
    - The function will check to make sure that the primary key of the new tuple does not match that of any other existing tuple within the relation.
    - If the new tuple passes the primary key check then it is inserted into the desired relation.

delete Function:

- **Description:** This function is a general delete function used to delete a tuple from any of the relations in the DB (Student, Course, School, etc).
- **Generalized Code:**

```
DELETE FROM TABLE_NAME
WHERE PKAttr = 'desiredPK';
```

- **Input:** The primary key of the tuple in the relation the user is wanting to delete from.
- **Steps:**
    - The user selects which relation they want to delete a tuple from.
    - The user will input the primary key of the tuple they wish to delete.

    - The function will check to see if this primary key exists in the relation the user is wanting to delete from.
    - If the primary key exists then the tuple is deleted, and any tuples with this primary key in other relations with dependencies are deleted as well since all tables that have referenced attributes have ON DELETE CASCADE added to them.

update Function:

- **Description:** This function is a general update function used to update a tuple in any of the relations in the DB (Student, Course, School, etc.)
- **Generalized Code:**

```
UPDATE TABLE_NAME
SET Column1=value1,...,ColumnN=valueN
WHERE PKAttr = 'desiredPK';
```

- **Input:** The primary key of the tuple that the user wants to update, the attributes in the tuple they want to update, as well as what they want to update them to.
- **Steps:**
    - The user selects which relation they want to update a tuple in.
    - The user will then input the primary key of the tuple they wish to update.
    - The function will check to make sure that the primary key of the tuple matches that of a tuple within the relation.
    - If the primary key check is passed, then the user can select which attributes they wish to update and input the updated information for the attributes in that tuple.
    - The tuple will then be updated within the relation once submitted as all relations which are referenced have ON UPDATE CASCADE added to them.

getTuple Function:

- **Description:** This function is to get all or specified information of a single tuple from any of the relations (Student, Faculty, School, etc.).

- **Generalized Code:**

```
SELECT Desired_Attributes
FROM TABLE_NAME
WHERE PKAttr = 'desiredPK';
```

- **Input:** Relation to get from, and the tuple's primary key.
- **Steps:**
    - The user is asked to input the relation they wish to get information from, and the primary key of the tuple they are looking for.
    - The user inputs the primary key, and the function checks to see if the relation contains said primary key.
    - If the primary key check is passed then the function will find and return the tuple from that relation and all the specified attributes from the relation.

getSchoolCourses Function:

- **Description:** This function will get all the courses offered at a particular school.
- **Generalized Code:**

```
SELECT DISTINCT CourseName
FROM COURSE
WHERE Course_School = 'School_Name';
```

- **Input:** School name.
- **Steps:**
    - The user will input the school name to find courses from.
    - The function will then access the "Courses" relation and return all unique course names with the Course_School equal to the entered school.

getMaxCredits Function:

- **Description:** This function will get the maximum credit hours out of all the courses offered at school and return the max credit hours along with the name of the course with the max credit hours.
- **Generalized Code:**

```
SELECT CourseName, CreditHours
FROM COURSE
WHERE Course_School = 'School_Name' AND CreditHours =
                                          (SELECT MAX(CreditHours)
                                           FROM COURSE
                                           WHERE Course_School = 'School_Name');
```

- **Input:** The name of the school to search for courses at.
- **Steps:**
    - The user will input the school name to find courses from.
    - The function will then first on the inner loop use the max aggregate function to find the value for the max CreditHours of courses with Course_School equal to the entered school.
    - The outer loop will then return the course name and credit hours of the course(s) with Course_School, and CreditHours equal to the max credit hours calculated in the inner loop.

getMinCredits Function:

- **Description:** This function will get the minimum credit hours out of all the courses offered at school and return the min credit hours along with the name of the course with the min credit hours.

- **Generalized Code:**

```
SELECT CourseName, CreditHours
FROM COURSE
WHERE Course_School = 'School_Name' AND CreditHours =
                                          (SELECT MIN(CreditHours)
                                           FROM COURSE
                                           WHERE Course_School = 'School_Name');
```

- **Input:** The name of the school to search for courses at.
- **Steps:**
    - The user will input the school name to find courses from.
    - The function will then first on the inner loop use the min aggregate function to find the value for the min CreditHours of courses with Course_School equal to the entered school.
    - The outer loop will then return the course name and credit hours of the course(s) with Course_School equal to the entered school name, and CreditHours equal to the min credit hours calculated in the inner loop.

countStudentsSchool Function:

- **Description:** This function will get the count of the number of students who are enrolled at a single school.
- **Generalized Code:**

```
SELECT COUNT(A.StudentID)
FROM ATTENDS AS A
WHERE A.School_Name = 'School_Name';
```

- **Input:** The name of the school to find the student count for.
- **Steps:**
    - The user will input the school name to find count students from.
    - The function will then access the “Attends” relation.

- It will then select only the tuples with School_Name equal to the entered school and then use the aggregate count function to count the total number of StudentID's from the remaining tuples.
- The function will then return the count of the total number of students who attend that particular school.

countStudentsCourse Function:

- **Description:** This function will get the count of the number of students in a particular course at a school.
- **Generalized Code:**

```
SELECT COUNT(T.StudentID)
FROM COURSE AS C, TAKES AS T
WHERE C.CourseID = T.CourseID AND C.CourseName = 'Course_Name' AND C.Course_School =
'School_Name';
```

- **Input:** School name and course name to count the number of students from.
- **Steps:**
  - The user will input the school and course name to count.
  - The function will then do a cartesian cross between the course and take tables, selecting only tuples with CourseID's equal to each other.
  - It will then select only tuples from that which have CourseName equal to the entered course name, and Course_School equal to the entered course school.
  - After this, it will then use the aggregate count function to count the number of StudentID's from the given course at the given school and return the count.

avgSchoolGPA Function:

- **Description:** This function will get the average GPA of all the students at a particular school for a particular semester.

- **Generalized Code:**

```
SELECT AVG(ACA.GPA)
FROM ATTENDS AS ATT, ACADEMIC_RECORD AS ACA
WHERE ATT.StudentID = ACA.StudentID AND ATT.School_Name = 'School_Name' AND
ACA.SemesterID = 'Semester_ID';
```

- **Input:** School name to get the average from, and SemesterID for the desired semester average.
- **Steps:**
    - The user will input the school name to get average from.
    - The function will then do a cartesian cross of the attends and academic_records tables, selecting only tuples with StudentID's equal to one another, with School_Name equal to the entered school name, and with SemesterIDs equal to the entered SemesterID.
    - The function will then average the GPAs of all the remaining tuples to give the average GPA at that school for the entered semester.

maxSchoolGPA Function:

- **Description:** This function will get the max GPA from a particular school for a given semester.
- **Generalized Code:**

```
SELECT MAX(ACA.GPA)
FROM ATTENDS AS ATT, ACADEMIC_RECORD AS ACA
WHERE ATT.StudentID = ACA.StudentID AND ATT.School_Name = 'School_Name' AND
ACA.SemesterID = 'Semester_ID';
```

- **Input:** School name to get max from, and SemesterID for the specific semester.
- **Steps:**
    - The user will input the school name to get max gpa from.

- The function will then do a cartesian cross of the attends and academic_records tables, selecting only tuples with StudentID's equal to one another, with School_Name equal to the entered school name, and with SemesterIDs equal to the entered SemesterID.
- The function will then get the max GPA from the school and return it.

minSchoolGPA Function:

- **Description:** This function will get the min GPA from a particular school for a given semester.
- **Generalized Code:**

```
SELECT MIN(ACA.GPA)
FROM ATTENDS AS ATT, ACADEMIC_RECORD AS ACA
WHERE ATT.StudentID = ACA.StudentID AND ATT.School_Name = 'School_Name' AND
ACA.SemesterID = 'Semester_ID';
```

- **Input:** School name to get min from, and SemesterID for the specific semester.
- **Steps:**
  - The user will input the school name to get min gpa from.
  - The function will then do a cartesian cross of the attends and academic_records tables, selecting only tuples with StudentID's equal to one another, with School_Name equal to the entered school name, and with SemesterIDs equal to the entered SemesterID.
  - The function will then get the min GPA from the school and return it.

getStudentCourses Function:

- **Description:** This function will return all the courses that a student is taking.

- **Generalized Code:**

```
SELECT C.CourseName
FROM COURSE AS C, TAKES AS T
WHERE C.CourseID = T.CourseID AND T.StudentID = 'Desired_StudentID';
```

- **Input:** The StudentID of the student to get courses for.
- **Steps:**
    - The user will input the StudentID for the desired student.
    - The function will then do a cartesian cross between the Course and Takes relationships, setting the CourseID's equal to one another, and only selecting tuples with StudentID equal to the entered StudentID.
    - The function will then return the course names for all the courses the entered student is taking.

getStudentClubs Function:

- **Description:** This function will return all the clubs that a student is a part of.
- **Generalized Code:**

```
SELECT Club_Name
FROM JOINS
WHERE StudentID = 'Desired_StudentID';
```

- **Input:** The StudentID of the student to get clubs for.
- **Steps:**
    - The user will input the StudentID for the desired student.
    - The function will then access the "Joins" relationship, and retrieve all the club names that the student is a part of using the entered StudentID.
    - The function will then return the club names for all the clubs retrieved in the prior step.

getStudentSchool Function:

- **Description:** This function will return the school that a particular student of interest attends.
- **Generalized Code:**

```
SELECT S.LastName, S.FirstName, A.School_Name
FROM ATTENDS AS A, STUDENT AS S
WHERE S.StudentID = A.StudentID AND A.StudentID = 'StudentID';
```

- **Input:** The StudentID of the student to get school for.
- **Steps:**
    - The user will input the StudentID for the desired student.
    - The function will then do a cartesian cross between the Attends and Student relations, setting the StudentID's equal to one another, and then select the tuple with the StudentID equal to the entered StudentID.
    - The function will then return the name of the school the student attends, as well as their first and last name.

calcGradCreds Function:

- **Description:** This function will calculate and return the number of credits a student has left before they graduate.
- **Generalized Code:**

```
SELECT 'Creds_To_Graduate' - SUM(Semester_Credits)
FROM ACADEMIC_RECORD
WHERE StudentID = 'Desired_StudentID';
```

- **Input:** StudentID and number of credits required for graduation.
- **Steps:**
    - The user will input the StudentID for the desired student, and the number of credits needed to graduate at the given school.

- The function will then access the “Academic_Records” relation, and select the tuples with StudentID equal to the entered StudentID.
- The function will then use the aggregate sum function on the Semester_Credits columns of the tuples selected in the prior step.
- The function will then subtract the sum of credits the student has from the credits required to graduate and then return the result.

getSchoolFaculty Function:

- **Description:** This function will get and return all the faculty members that work at a particular school.
- **Generalized Code:**

```
SELECT F.Faculty_Name
FROM SCHOOL AS S, WORKS AS W, FACULTY AS F
WHERE S.School_Name = W.School_Name AND W.FacultyID = F.FacultyID AND S.School_Name =
'School_Name';
```

- **Input:** The School name to get faculty list for.
- **Steps:**
  - The user will input the school name to get a faculty member list for.
  - The function will then do a cartesian cross between the School, Works, and Faculty relations, setting the School_Name between the School and Works relations equal to one another, and the FacultyID between the Works and Faculty relations equal to one another.
  - Of the tuples remaining from the prior cartesian cross, the function will then select tuples from them with the School_Name equal to that of the entered school name.
  - The function will then return all the faculty members names of the remaining tuples.

getFacultyCourses Function:

- **Description:** This function will return all the courses that a faculty member is currently teaching.
- **Generalized Code:**

```
SELECT C.CourseName
FROM TEACHES AS T, COURSE AS C
WHERE C.CourseID = T.CourseID AND T.FacultyID = 'Desired_FacultyID';
```

- **Input:** The FacultyID of the faculty member to get courses for.
- **Steps:**
    - The user will input the FacultyID for the desired member.
    - The function will then do a cartesian cross of the Teaches and Course relations, setting the CourseID's equal to one another.
    - Of the tuples remaining from the cartesian cross in the prior step, the function will then only select tuples that have a FacultyID equal to the entered FacultyID.
    - The function will then display the course names of the remaining tuples from the above step.

cumulativeGPA Function:

- **Description:** This function is used to get the cumulative gpa of a student by averaging their past semester GPAs.
- **Generalized Function:**

```
SELECT FORMAT(AVG(GPA), 4)
FROM ACADEMIC_RECORD
WHERE StudentID = 'Desired_StudentID';
```

- **Input:** The StudentID the user wishes to get the cumulative GPA for.
- **Steps:**

- The user will input the StudentID for the desired student.
- The function will then use the aggregate function on the “Academic_Record” relation to get the average of the semester GPAs of that student to represent the cumulative GPA.
- The function will then return the calculated cumulative GPA from the prior step with a precision of 4 decimal places.

clubMemberCount Function:

- **Description:** This function is used to get the member count of a given club at a school.
- **Generalized Code:**

```
SELECT COUNT(StudentID)
FROM JOINS
WHERE Club_Name = 'Desired_Club_Name' AND Club_School = 'Desired_School_Name';
```

- **Input:** The club name and the school name for that club.
- **Steps:**
  - The user will input the club name and school name of the club they want the member count of.
  - The function will then select only tuples with Club_Name and School_Name equal to that of the entered club name and school name.
  - The function will then do an aggregate count function on the StudentID’s of the remaining tuples.
  - The function will then return the counted members in the club.

getCoursePeriod Function:

- **Description:** This function is used to get the period of which a specific course at a specific school happens during.

- **Generalized Code:**

```
SELECT CP.ClassPeriod
FROM COURSE AS CO, CLASS_PERIOD AS CP
WHERE CO.CourseID = CP.CourseID AND CO.CourseName = 'Course_Name' AND CO.Course_School
= 'School_Name';
```

- **Input:** The course name and school name that the course is offered at.
- **Steps:**
    - The user will input the course name and school name for the desired course.
    - The function will then do a cartesian cross of the course and class period tables by setting the CourseID's in both tables equal to one another.
    - The function will then select only tuples with the entered course name and school name and return the class period they are offered during.

LOGICAL DATABASE DESIGN:

Attribute Table:

Attribute	Meaning	Data Type	Integrity Constraints
StudentID	ID number of a Student	Integer	Entity integrity - StudentID is a primary key which cannot be NULL Referential integrity - Academic_Record's StudentID references Student's StudentID; Student_Emergency_Contact entity references Student's StudentID
DateOfBirth	Birthday of a Student	Date	Not Null
GradeLevel	Grade level of a Student	Smallint	Not Null
StudentEmail	Email address of a	Varchar	

	Student		
FullName	Consists of first, middle, and last name of a Student	Varchar	Not Null
FirstName	First name of a Student	Varchar	Not Null
MiddleName	Middle name of a Student	Varchar	
LastName	Last name of a Student	Varchar	Not Null
Credits	Amount of credits a Student has taken	Smallint	
CumulativeGPA	A Student's cumulative GPA	Float	Not Null
Semester	A semester which a Student has taken credits in	Varchar	Not Null
GPA	A Student's GPA corresponding to a semester	Float	Not Null
EC_Name	Name of a Student Emergency Contact	Varchar	Not Null
Relationship	The relationship between a Student Emergency Contact and its corresponding Student (ex. parent, sibling, etc.)	Varchar	
EC_Phone	Phone number of a Student Emergency Contact	Char	Not Null
EC_Email	Email address of a Student Emergency Contact	Varchar	Not Null
School_Name	Name of a School	Varchar	Not Null
Address	Address of a School	Varchar	Entity integrity - Address is a primary key which cannot be

			NULL
School_Phone	Phone number of a School	Char	Not Null
Faculty_Name	Name of a faculty member	Varchar	Not Null
FacultyID	ID number of a Faculty member	Integer	Entity integrity - FacultyID is a primary key which cannot be NULL
Salary	Salary of a Faculty member	Float	Not Null
StartDate	The date that a Faculty member started working at a School	Date	Not Null
Department	The department a faculty member works in	Varchar	Not Null
CourseID	Id number of a Course	Integer	Entity integrity - CourseID is a primary key which cannot be NULL
CourseName	Name of a Course	Varchar	Not Null
ClassPeriod	Class period(s) a Course takes place in	Smallint	
RoomNumber	Number of the classroom a Course takes place in	Smallint	
Course_School	School that a Course is offered at	Varchar	Not Null
CreditHours	Number of credit hours a Course is worth	Smallint	
Club_School	School a Club takes place at	Varchar	Entity integrity - ClubSchool is a primary key which cannot be NULL
MemberCount	Number of members	Smallint	Not Null

	in a Club		
Club_Name	Name of a Club	Varchar	Entity integrity - ClubName is a primary key which cannot be NULL
Sponsor	Sponsor of a Club	Varchar	Not Null
LeaderID	ID of student leader of a Club	Varchar	

Phase 2 step 3 (I left this page here for easy copy and pasting since the page below is an image)

Student(StudentID, FirstName, MiddleName, LastName, DateOfBirth, GradeLevel, StudentEmail)

Student_Emergency_Contact(StudentID, EC_Name, Relationship)

Emergency_Phone_Number(StudentID, EC_Name, EC_Phone)

Emergency_Email(StudentID, EC_Name, EC_Email)

Academic_Record(StudentID, SemesterID, Semester_Credits, GPA)

School(Address, School_Name, PhoneNumber)

Clubs(Club_Name, Club_School, Sponsor, LeaderID)

Joins(StudentID, Club_Name, Club_School)

Attends(StudentID, School_Name)

Course(CourseID, CourseName, RoomNumber, Course_School, CreditHours)

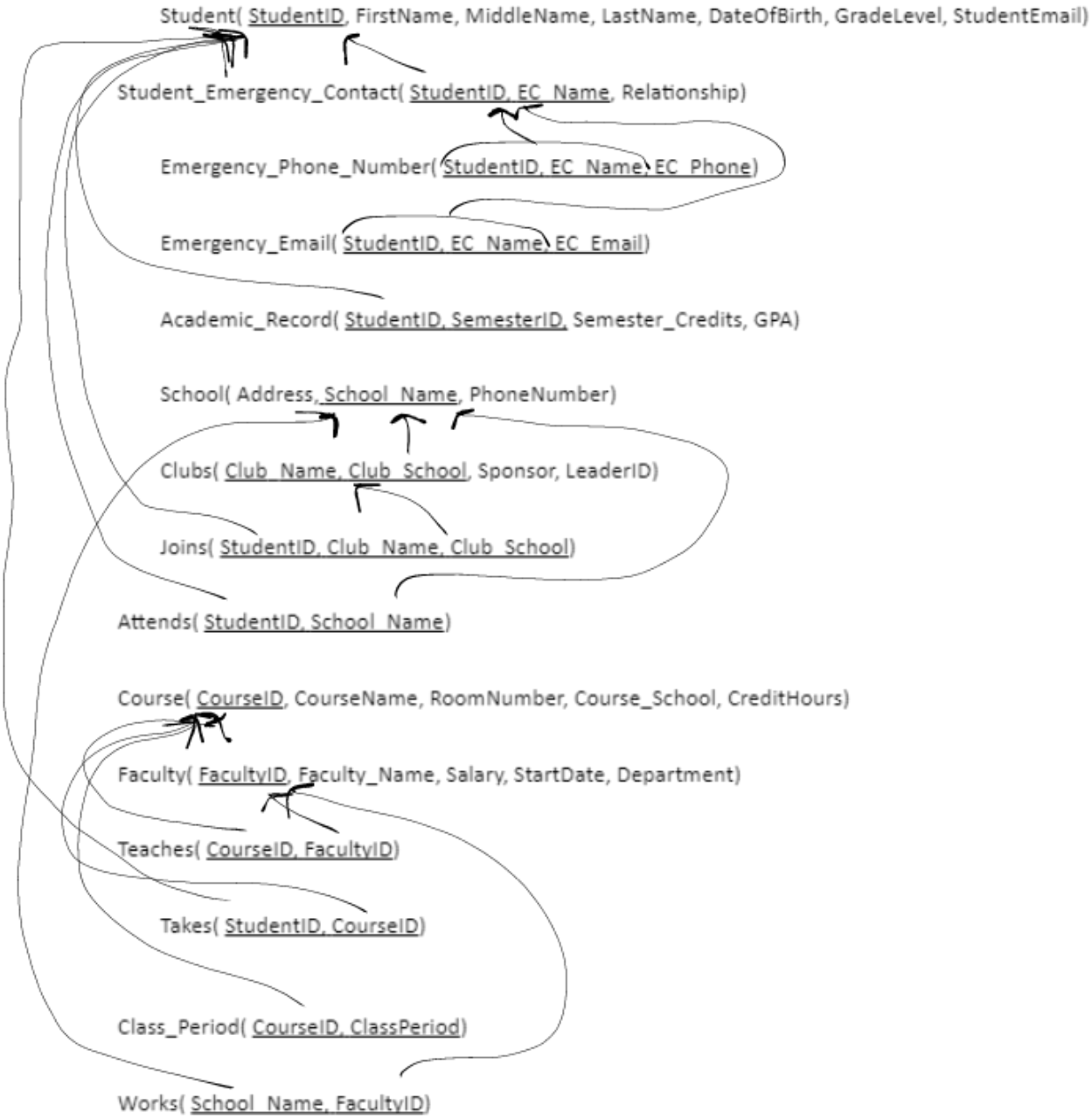
Faculty(FacultyID, Faculty_Name, Salary, StartDate, Department)

Teaches(CourseID, FacultyID)

Takes(StudentID, CourseID)

Class_Period(CourseID, ClassPeriod)

Works(School_Name, FacultyID


Student(StudentID, FirstName, MiddleName, LastName, DateOfBirth, GradeLevel, StudentEmail)
Student_Emergency_Contact(StudentID, EC_Name, Relationship)
Emergency_Phone_Number(StudentID, EC_Name, EC_Phone)
Emergency_Email(StudentID, EC_Name, EC_Email)
Academic_Record(StudentID, SemesterID, Semester_Credits, GPA)
School(Address, School_Name, PhoneNumber)
Clubs(Club_Name, Club_School, Sponsor, LeaderID)
Joins(StudentID, Club_Name, Club_School)
Attends(StudentID, School_Name)
Course(CourseID, CourseName, RoomNumber, Course_School, CreditHours)
Faculty(FacultyID, Faculty_Name, Salary, StartDate, Department)
Teaches(CourseID, FacultyID)
Takes(StudentID, CourseID)
Class_Period(CourseID, ClassPeriod)
Works(School_Name, FacultyID)

School Database User Manual:

Important Beginning Information:

It is important to specify that for all these functions, the user must input them into the DBMS of their choosing, in our case we used myphpAdmin in tandem with XAMPP, in order to implement these functions. The functions should be input into the area in the DBMS that allows for querying.

Database Functions:

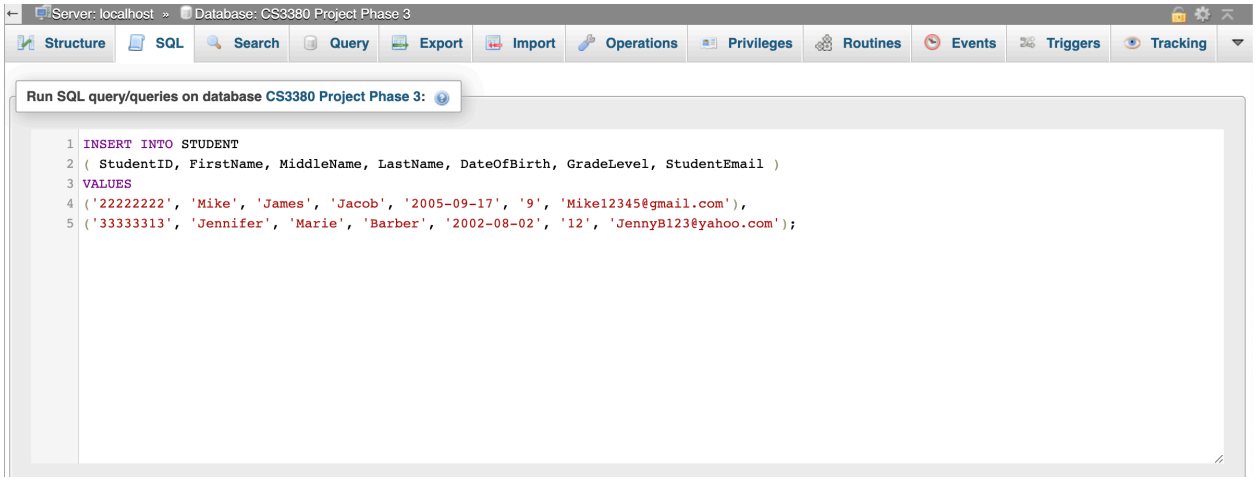
insert Function:

- **Description:** This function is a general insert function used to insert a new tuple into any of the relations in the DB (Student, Course, School, etc.)
- **Generalized Code:**

```
INSERT INTO TABLE_NAME
(Attr_1, Attr_2, Attr_3, ... , Attr_n)
VALUES
(T1A_1, T1A_2, T1A_3, ..., T1A_n),
(T2A_1, T2A_2, T2A_3, ..., T2A_n),
(T3A_1, T3A_2, T3A_3, ..., T3A_n),
...
(TnA_1, TnA_2, TnA_3, ..., TnA_n);
```

- **Input and Usage:** In this function the user will put in the name of the table "TABLE_NAME", attributes they would like to insert values for in a new tuple (Attr_1, Attr_2, Attr_3, ... , Attr_n), and the values they want to put in for the attributes of the entry they wish to insert (TA_1, TA_2, TA_3, ..., TA_n). The user should input the values between parenthesis with each value separated by a comma. Also each entry in parenthesis should be separated with a comma in the input. Attributes for each tables can be found in the relation section of the documentation.

- **Insert Function Example:**



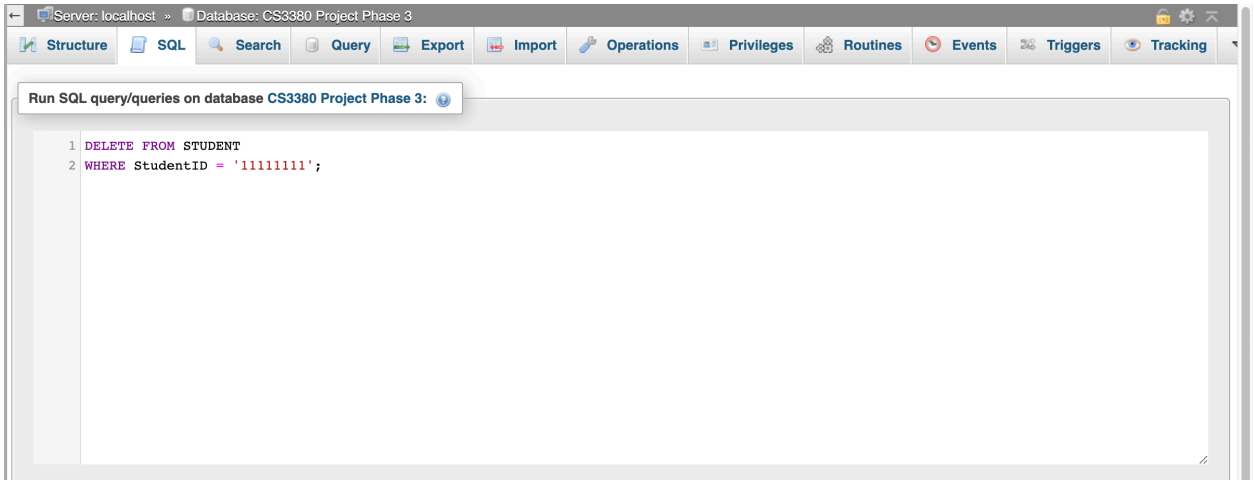
delete Function:

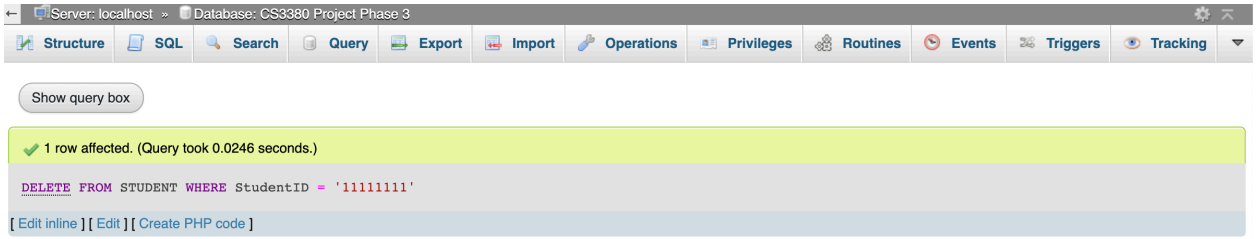
- **Description:** This function is a general delete function used to delete a tuple from any of the relations in the DB (Student, Course, School, etc).
- **Generalized Code**

```
DELETE FROM TABLE_NAME
WHERE PKAttr = 'desiredPK';
```

- **Input and Usage:** In this function the user will input the name of the table they wish to delete from “TABLE _NAME” and also input the name of the primary key attribute of that table “PKAttr” and the specific primary key of the entry they wish to delete from that table “desiredPK”. Note that the user should not have to worry about entries for deleted entities that are referenced in other tables as they are removed upon deletion of the referenced entity.

- **delete Function Example:**





update Function:

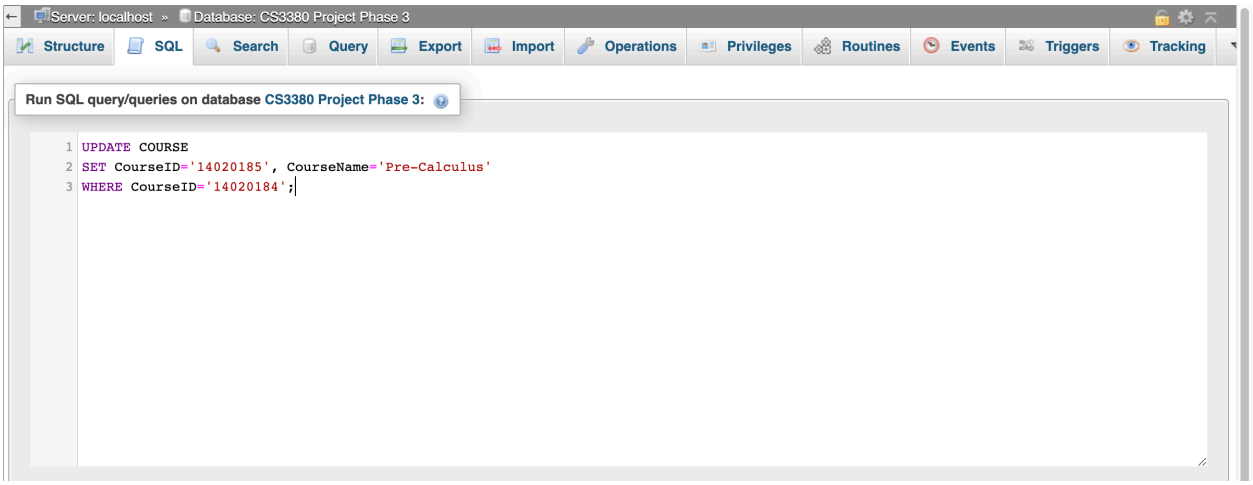
- **Description:** This function is a general update function used to update a tuple in any of the relations in the DB (Student, Course, School, etc.).
- **Generalize Code:**

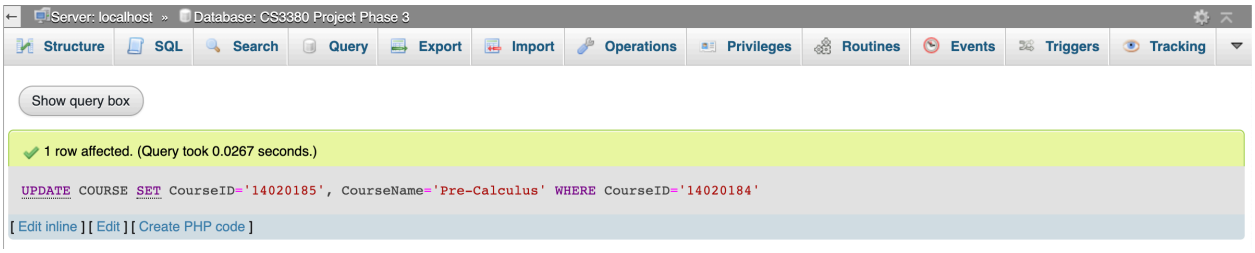
```
UPDATE TABLE_NAME
SET Column1=value1,...,ColumnN=valueN
WHERE PKAttr = 'desiredPK';
```

- **Input and Usage:** In this function the user will input the name of the table for which they wish to update a tuple within “TABLE_NAME”, the attributes for which they wish to change and the values they wish to change them to “Column1=value1,...,ColumnN=valueN”, and the primary key attribute of the table along with the specific primary key of the tuple they wish to update “PKAttr” and “desiredPK”. Note that like the delete function, the user does not need to worry about updating

referenced values in other tables if they are changed, this will occur automatically. The primary key attributes for each table can be found underlined in the relations section of the documentation.

- **update Function Example:**





getTuple Function:

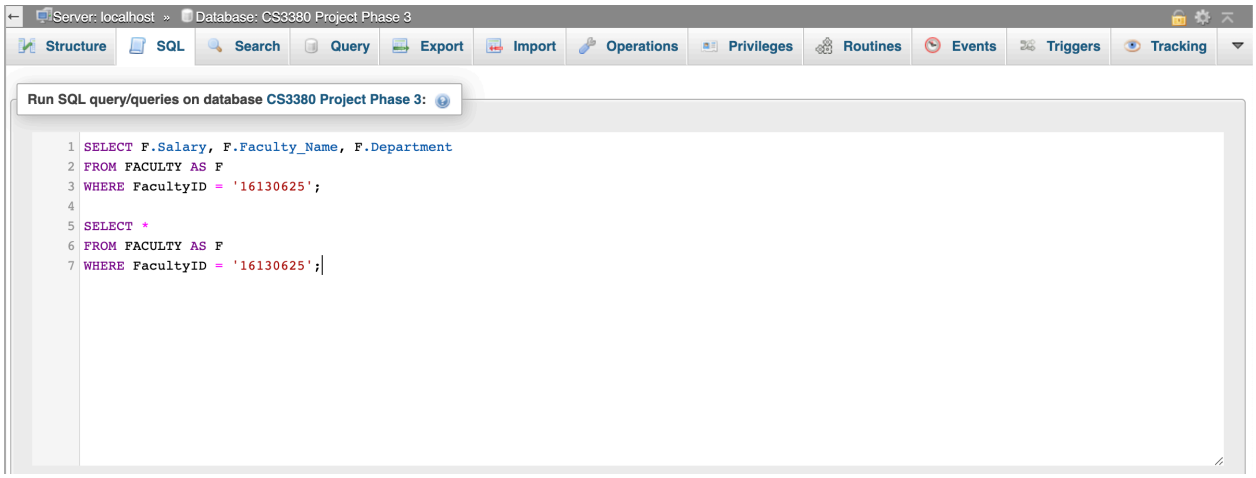
- **Description:** This function is to get all the information of a single tuple from any of the entity sets (Student, Faculty, School, etc.).
- **Generalized Code:**

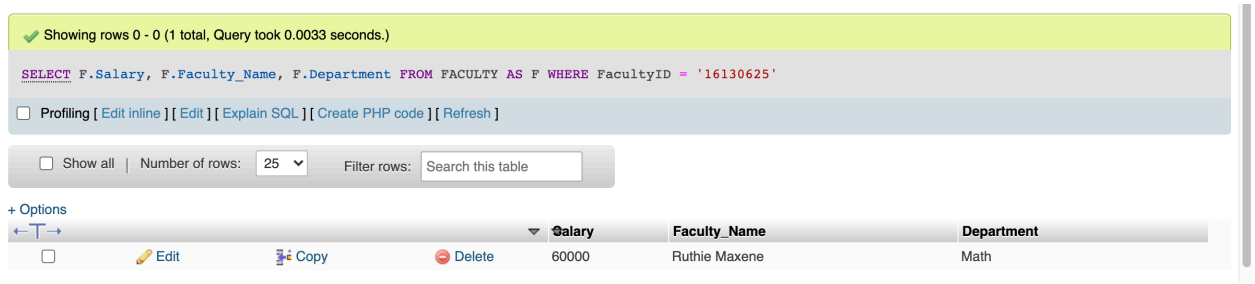
```
SELECT Desired_Attributes
FROM TABLE_NAME
WHERE PKAttr = 'desiredPK';
```

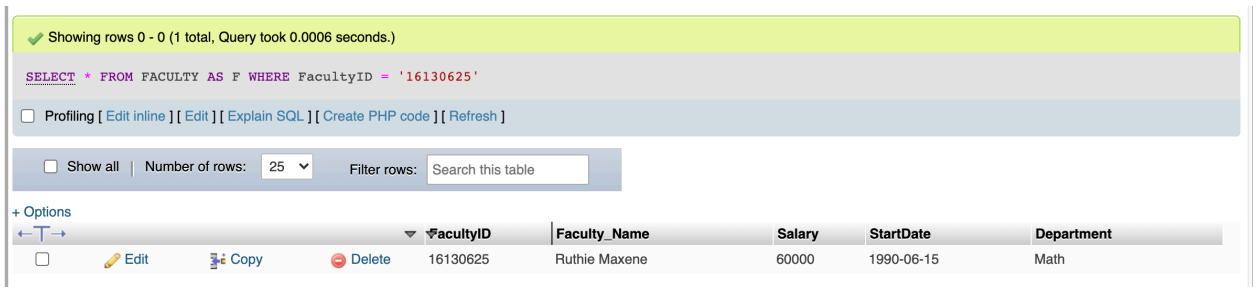
- **Input and Usage:** In this function the user should input the attributes they wish to get for a particular tuple “Desired_Attributes”, the name of the table the specific tuple is in “TABLE_NAME”, and the primary key attribute as well as the specific primary key for the

tuple they wish to get information for “PKAttr” and “desiredPK”. Note that if the user wishes to get the information of all attributes for the tuple they need only put an asterisk (*) in place of where the desired attributes are put. Primary key values can be found in the relations part of the documentation and are underlined for each relation.

- **getTuple Function Example:**







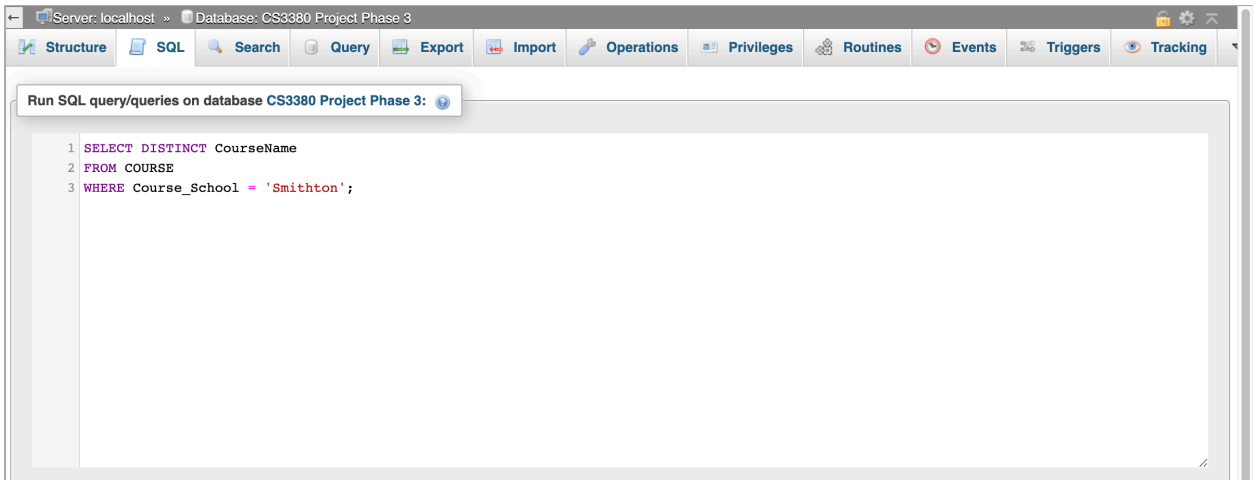
getSchoolCourses Function:

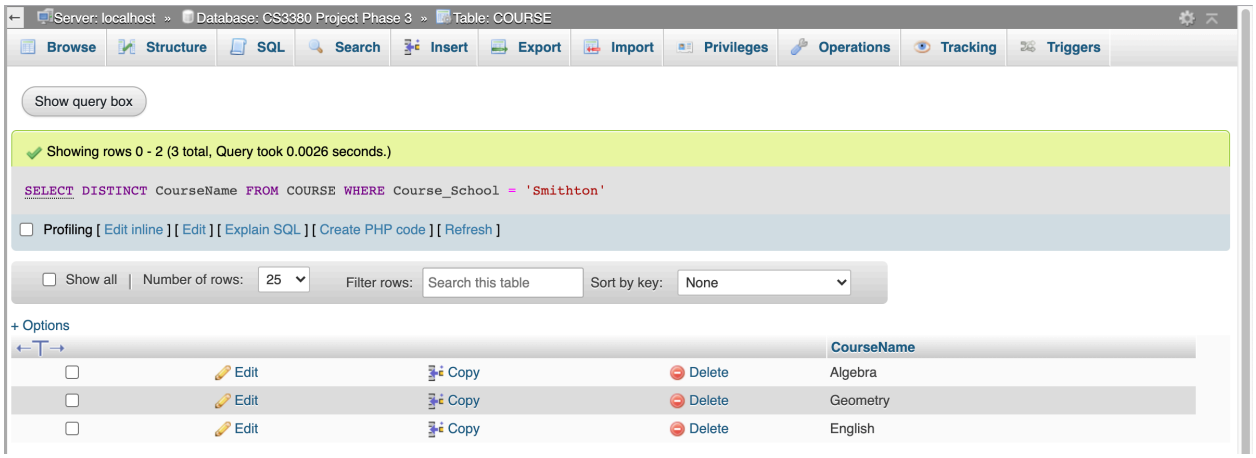
- **Description:** This function will get all the courses offered at a particular school.

- **Generalized Code:**

```
SELECT DISTINCT CourseName
FROM COURSE
WHERE Course_School = 'School_Name';
```

- **Input and Usage:** In this function the user needs to only input the name of the school for which they wish to get the courses from “School_Name”. Remember to make sure to input the school exactly as it was input into the database to get the function to return correctly.
- **getSchoolCourses Function Example:**





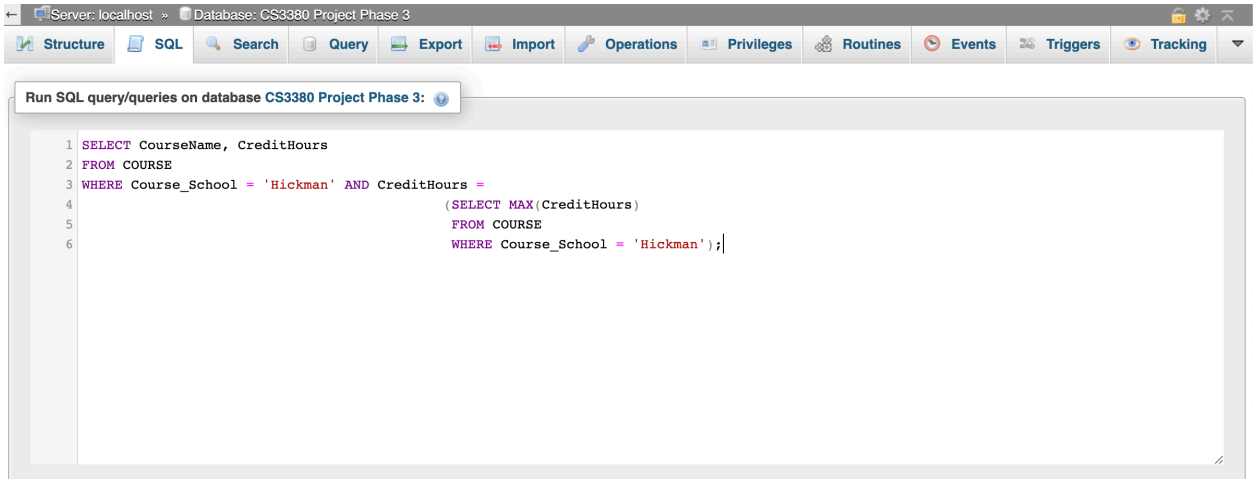
getMaxCredits Function:

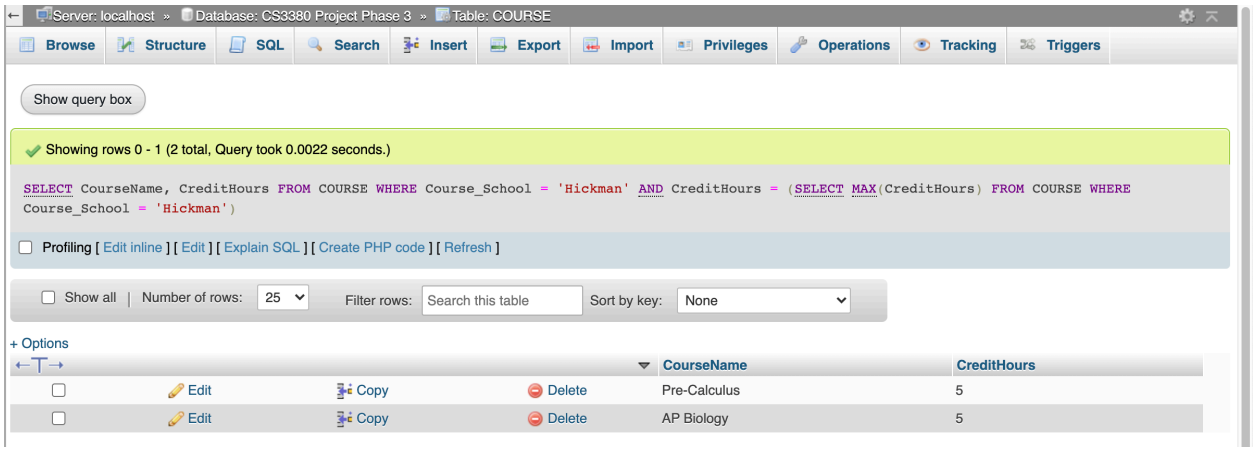
- **Description:** This function will get the maximum credit hours out of all the courses offered at a school and return the max credit hours along with the name of the course with the max credit hours.
- **Generalized Code:**

```
SELECT CourseName, CreditHours
FROM COURSE
WHERE Course_School = 'School_Name' AND CreditHours =
                                                (SELECT MAX(CreditHours)
                                                 FROM COURSE
                                                 WHERE Course_School = 'School_Name');
```

- **Input and Usage:** For this function, the user will simply input the name of the school they wish to get the course from “School_Name” and the function will return the course with the maximum credits. If there are 2 or more courses with the same max number of credit hours the function will return all of them.

- **getMaxCredits Function Example:**



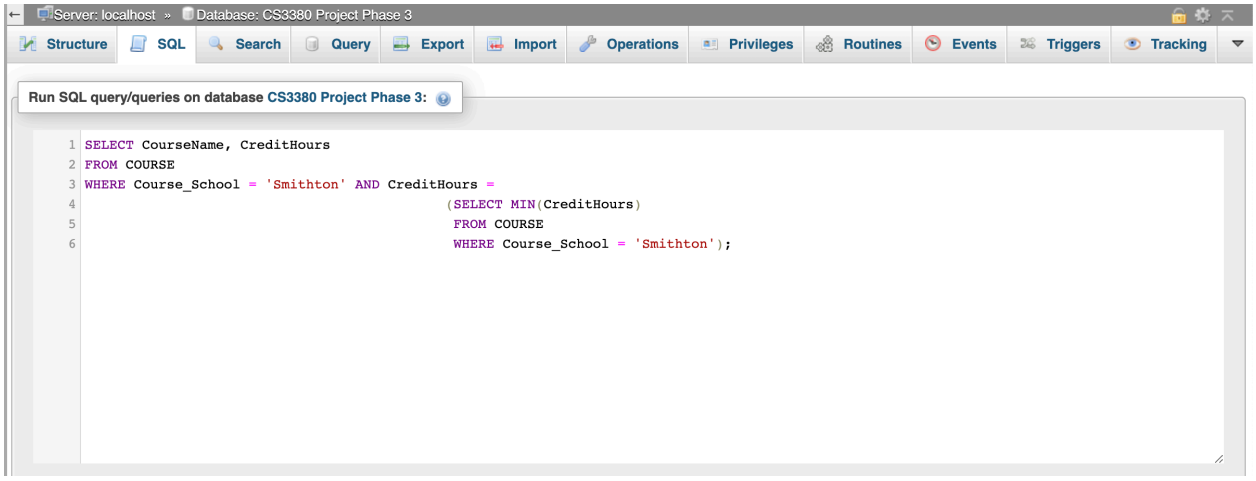


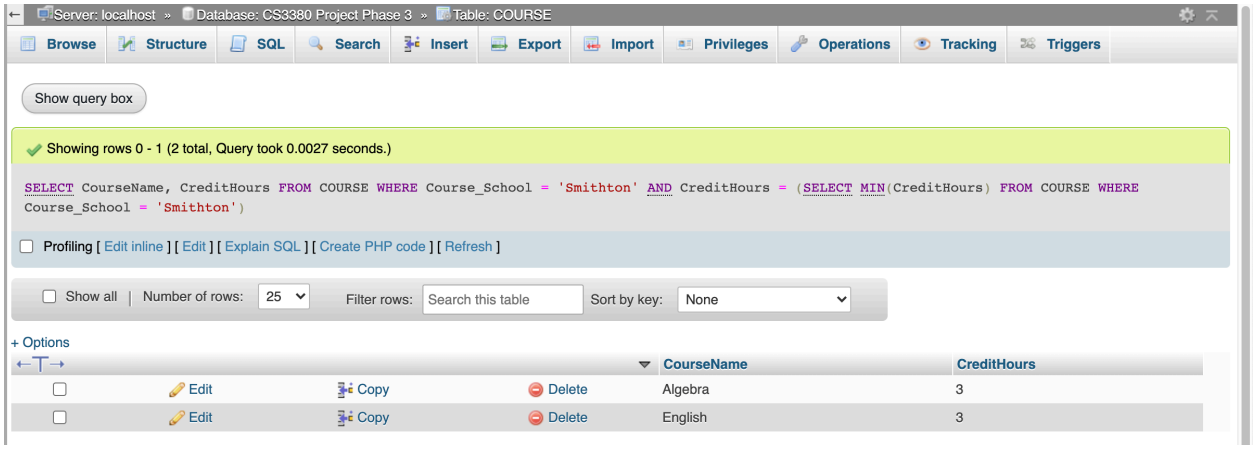
getMinCredits Function:

- **Description:** This function will get the minimum credit hours out of all the courses offered at a school and return the minimum credit hours along with the name of the course with the min credit hours.
- **Generalized Code:**

```
SELECT CourseName, CreditHours
FROM COURSE
WHERE Course_School = 'School_Name' AND CreditHours =
                                        (SELECT MIN(CreditHours)
                                         FROM COURSE
                                         WHERE Course_School = 'School_Name');
```

- **Input and Usage:** For this function, the user will simply input the name of the school they wish to get the course from “School_Name” and the function will return the course with the minimum credits. If there are 2 or more courses with the same min number of credit hours the function will return all of them.
- **getMinCredits Function Example:**





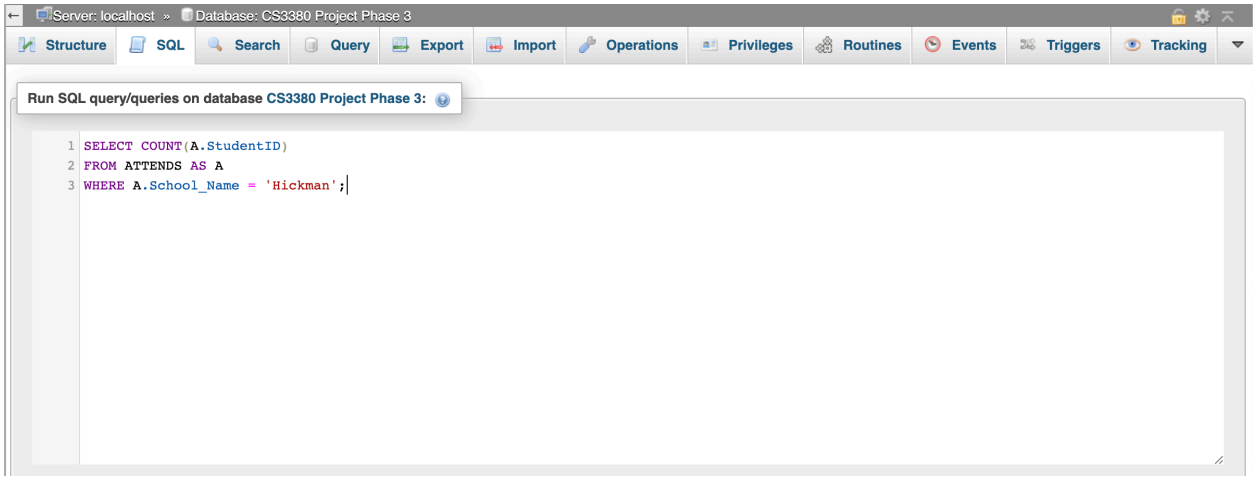
countStudentsSchool Function:

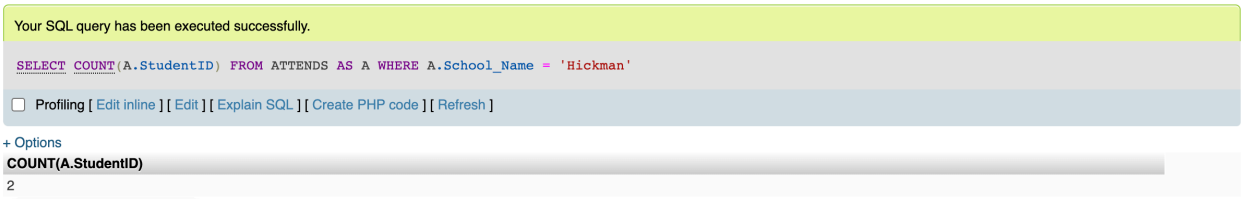
- **Description:** This function will get the count of the number of students who are enrolled at a single school.

- **Generalized Code:**

```
SELECT COUNT(A.StudentID)
FROM ATTENDS AS A
WHERE A.School_Name = 'School_Name';
```

- **Input and Usage:** In this function the user needs only to input the name of the school for which they wish to get the student count from in “School_Name”. It will return a single number for the count.
- **countStudentsSchool Function:**



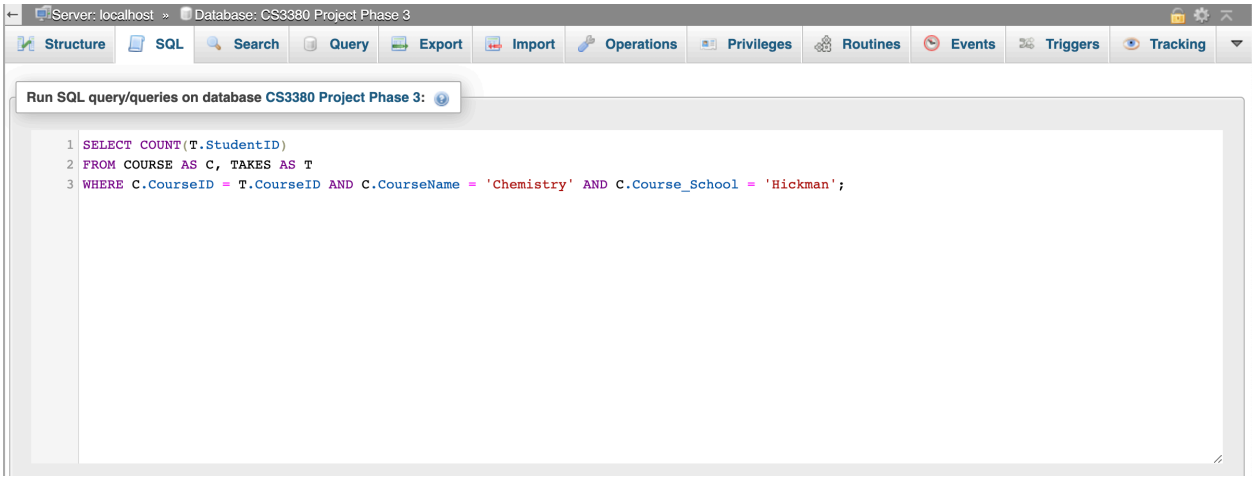


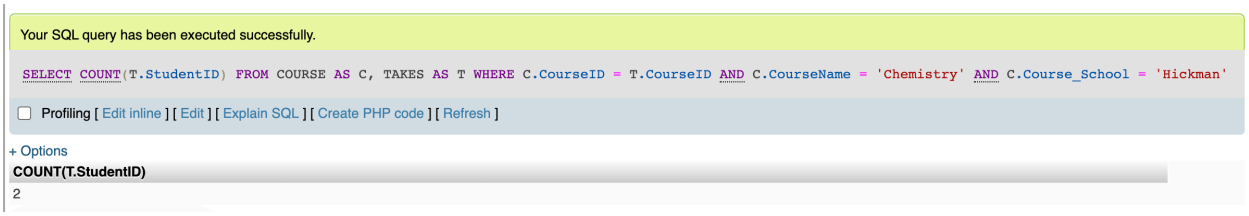
countStudentsCourse Function:

- **Description:** This function will get the count of the number of students in a particular course at a school.
- **Generalized Code:**

```
SELECT COUNT(T.StudentID)
FROM COURSE AS C, TAKES AS T
WHERE C.CourseID = T.CourseID AND C.CourseName = 'Course_Name' AND C.Course_School =
'School_Name';
```

- **Input and Usage:** For this function, the user must input the name of the course “Course_Name” and the name of the school with which the course is offered at “School_Name”. Note that the course name, like the school name must be put in exactly the same as it was first input into the database.
- **countStudentsCourse Function Example:**





avgSchoolGPA Function:

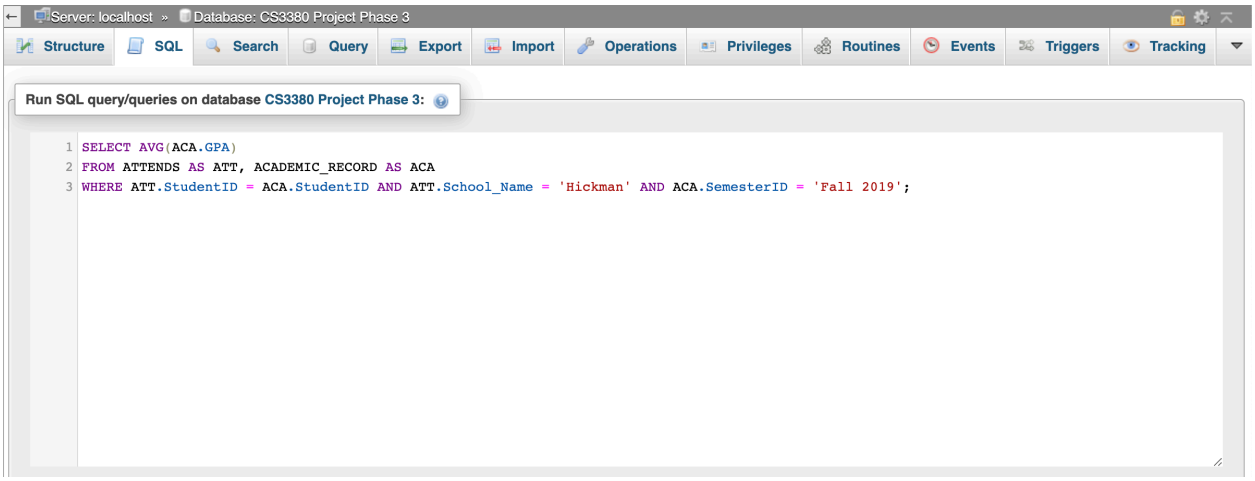
- **Description:** This function will get the average GPA of all the students at a particular school for a particular semester.
- **Generalized Code:**

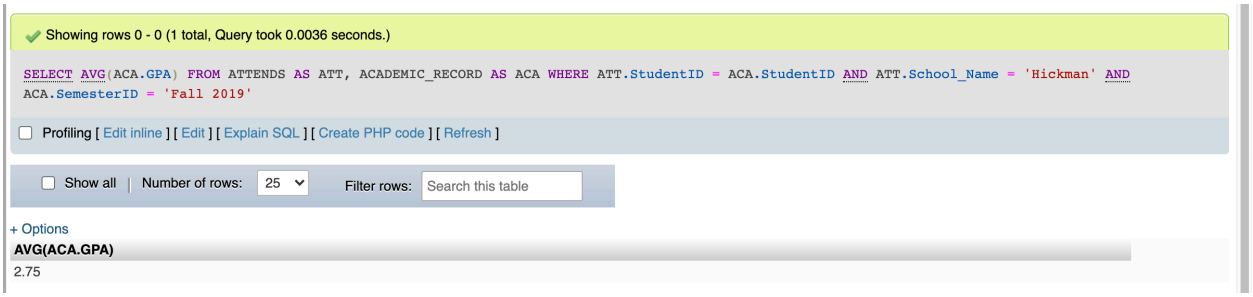
```
SELECT AVG(ACA.GPA)
FROM ATTENDS AS ATT, ACADEMIC_RECORD AS ACA
WHERE ATT.StudentID = ACA.StudentID AND ATT.School_Name = 'School_Name' AND
ACA.SemesterID = 'Semester_ID';
```

- **Input and Usage:** In this function the user must input the name of the school for which they wish to get the average gpa for “School_Name” and which semester gpa’s they

wish to select to get the average for “Semester_ID”. Note that like school, the SemesterID must be input into the function exactly as it was first entered into the database.

- **avgSchoolGPA Function Example:**





maxSchoolGPA Function:

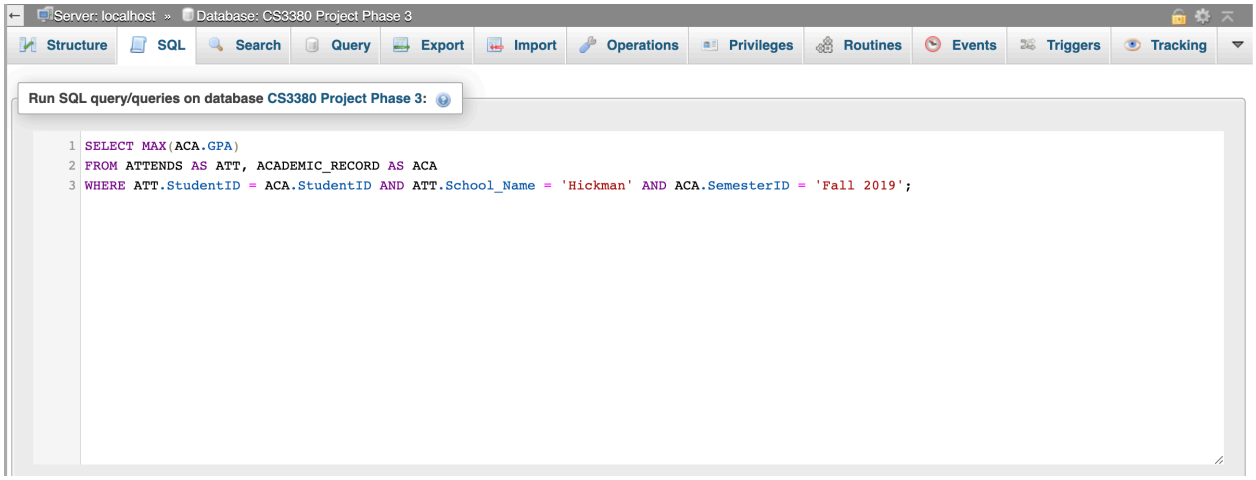
- **Description:** This function will get the max GPA from a particular school for a particular semester.
- **Generalized Code:**

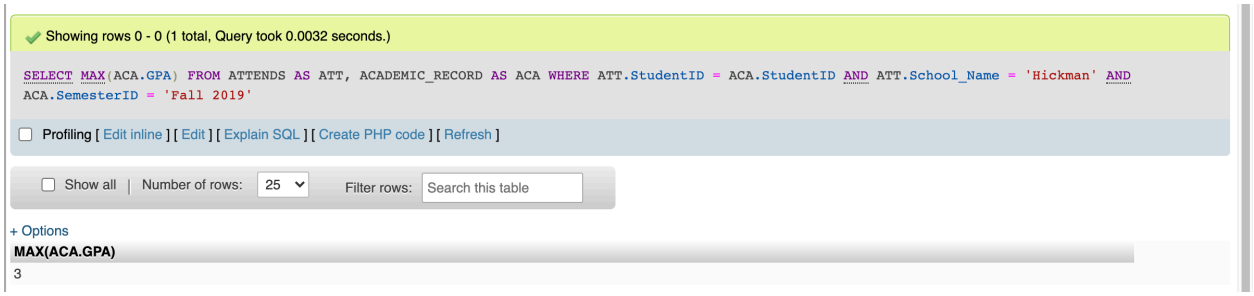
```
SELECT MAX(ACA.GPA)
FROM ATTENDS AS ATT, ACADEMIC_RECORD AS ACA
WHERE ATT.StudentID = ACA.StudentID AND ATT.School_Name = 'School_Name' AND
ACA.SemesterID = 'Semester_ID';
```

- **Input and Usage:** For this function, the user must input the school’s name “School_Name” and semester “SemesterID” they wish to select the max gpa from. Note

that both School_Name and SemesterID must be input into the function exactly as they were originally input into the database.

- **maxSchoolGPA Function Example:**





minSchoolGPA Function:

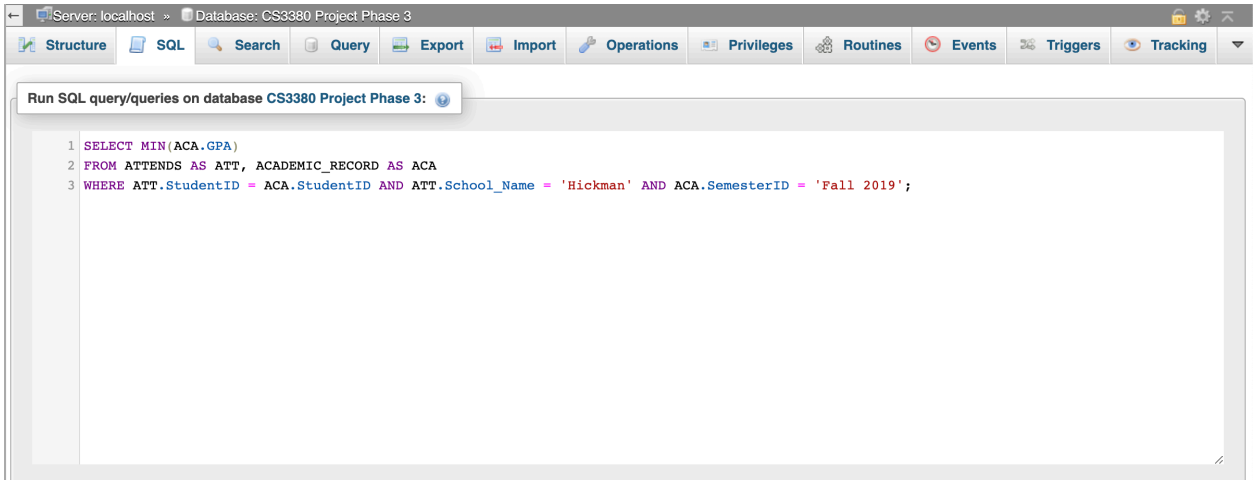
- **Description:** This function will get the min GPA from a particular school for a particular semester.
- **Generalized Code:**

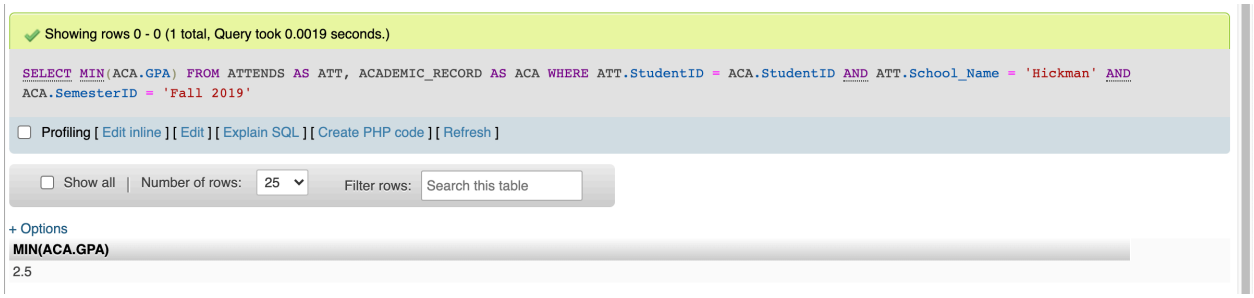
```
SELECT MIN(ACA.GPA)
FROM ATTENDS AS ATT, ACADEMIC_RECORD AS ACA
WHERE ATT.StudentID = ACA.StudentID AND ATT.School_Name = 'School_Name' AND
ACA.SemesterID = 'Semester_ID';
```

- **Input and Usage:** For this function, the user must input the school's name "School_Name" and semester "SemesterID" they wish to select the min gpa from. Note

that both School_Name and SemesterID must be input into the function exactly as they were originally input into the database.

- **minSchoolGPA Function Example:**





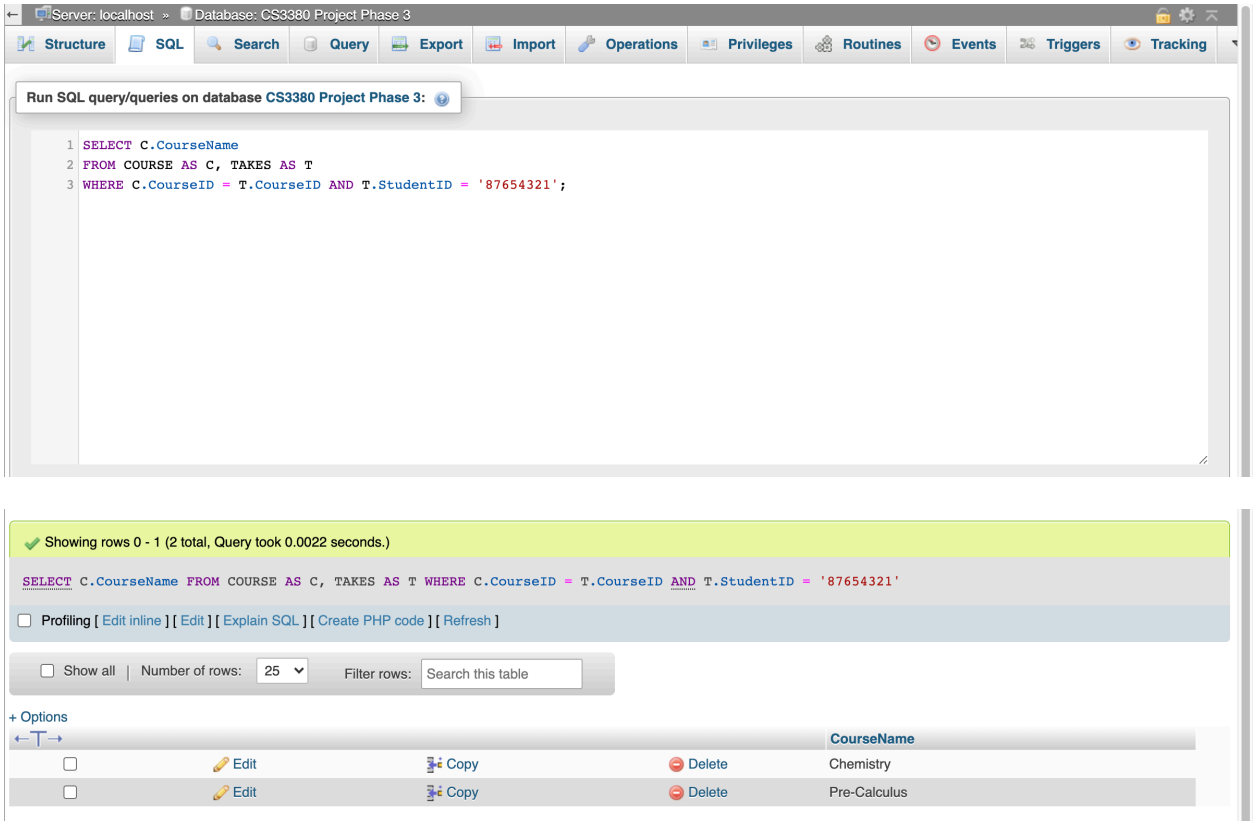
getStudentCourses Function:

- **Description:** This function will return all the courses that a student is currently taking.
- **Generalized Code:**

```
SELECT C.CourseName
FROM COURSE AS C, TAKES AS T
WHERE C.CourseID = T.CourseID AND T.StudentID = 'Desired_StudentID';
```

- **Input and Usage:** For this function, all the user must input is the StudentID in "Desired_StudentID" for the student of which they wish to get enrolled courses for.

- **getStudentCourses Function Example:**



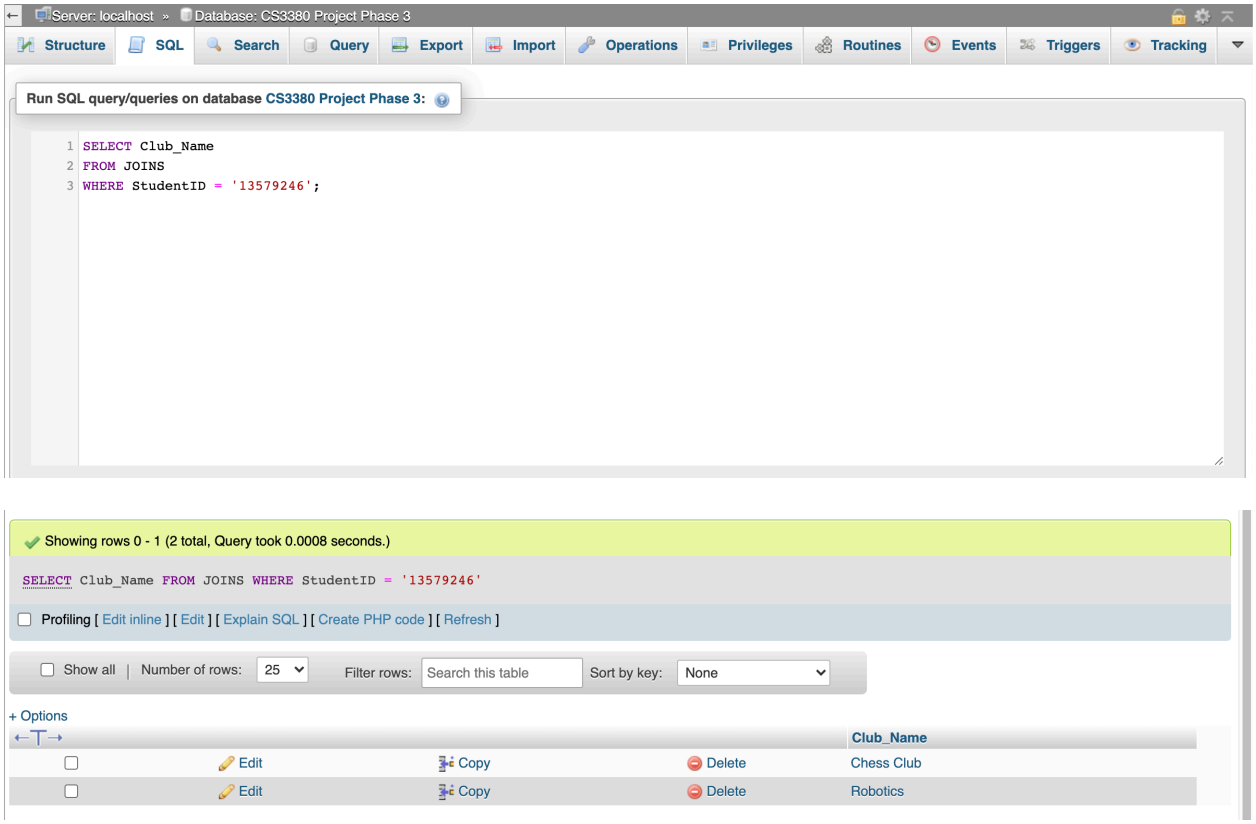
getStudentClubs Function:

- **Description:** This function will return all the clubs that a student is a part of.
- **Generalized Code:**

```
SELECT Club_Name
FROM JOINS
WHERE StudentID = 'Desired_StudentID';
```

- **Input and Usage:** For this function, the only thing the user must put in is the StudentID labeled “Desired_StudentID” in the code above of the student for which they wish to get the clubs for.

- **getStudentClubs Function Example:**



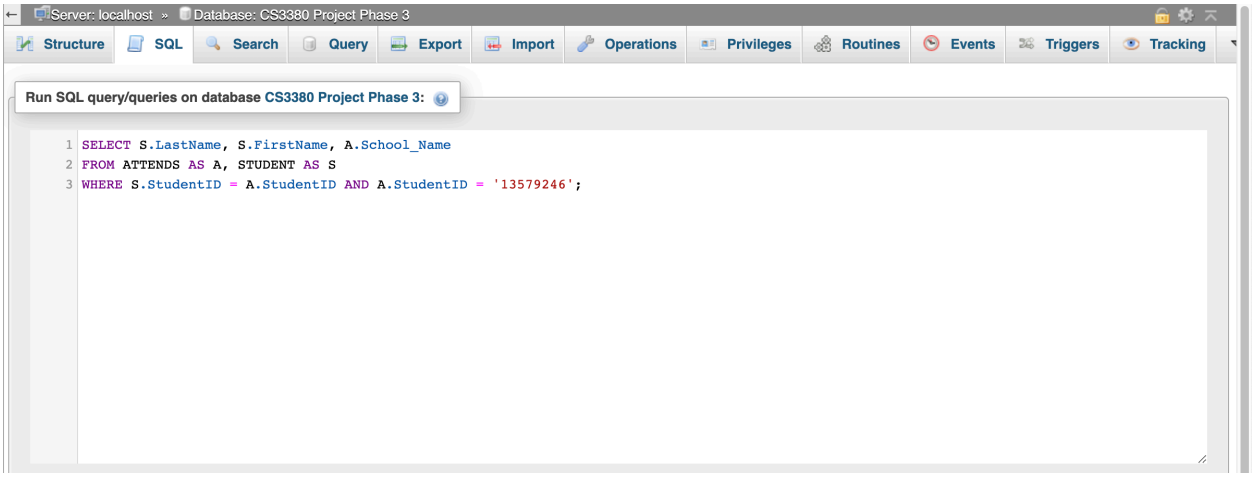
getStudentSchool Function:

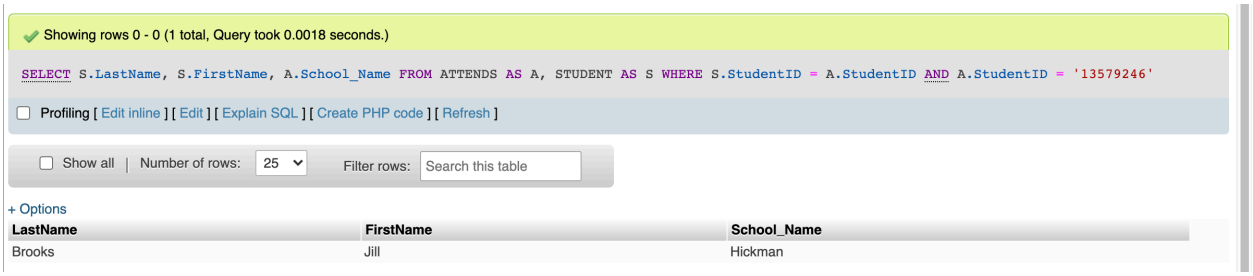
- **Description:** This function will return the school that a particular student of interest attends.
- **Generalized Code:**

```
SELECT S.LastName, S.FirstName, A.School_Name
FROM ATTENDS AS A, STUDENT AS S
WHERE S.StudentID = A.StudentID AND A.StudentID = 'StudentID';
```

- **Input and Usage:** For this function, the user simply needs to put in the “StudentID” of the student they wish to get the school for.

- **getStudentSchool Function Example:**





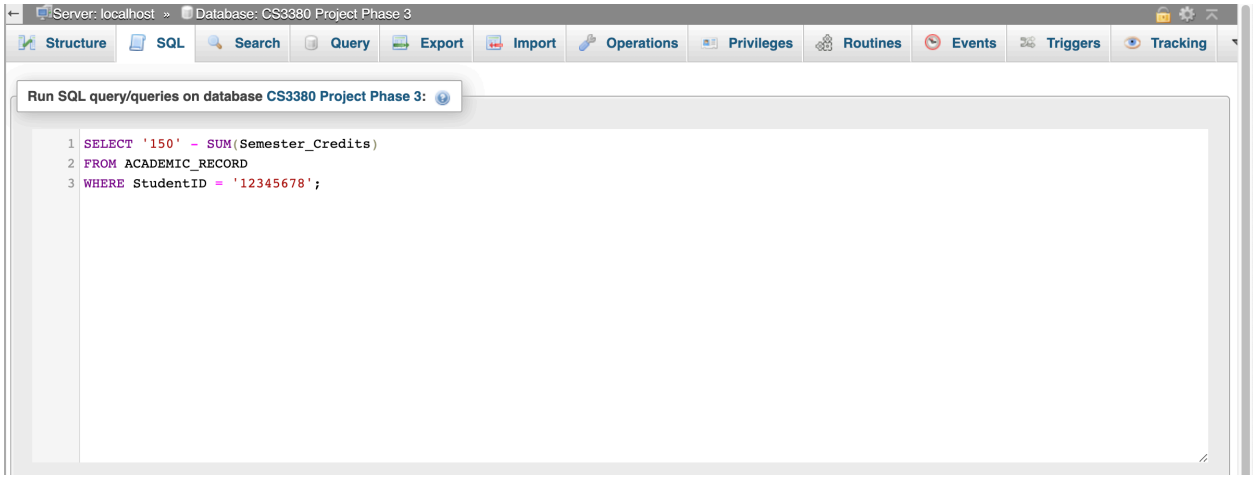
calcGradCreds Function:

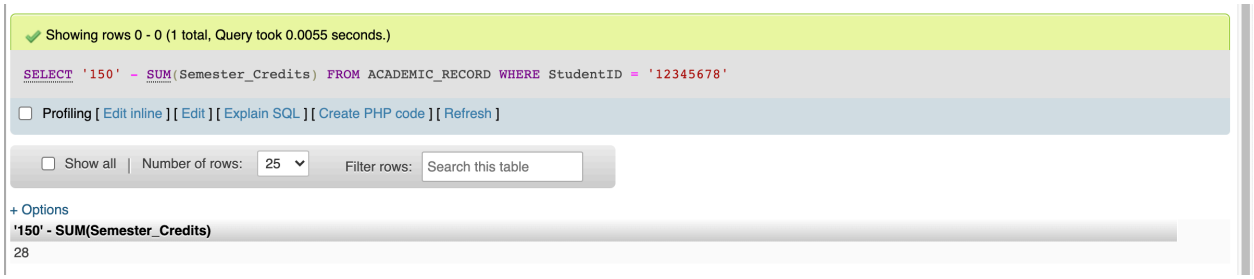
- **Description:** This function will calculate and return the number of credits a student has left that they need to get before they graduate.
- **Generalized Code:**

```
SELECT 'Creds_To_Graduate' - SUM(Semester_Credits)
FROM ACADEMIC_RECORD
WHERE StudentID = 'Desired_StudentID';
```

- **Input and Usage:** In this function, the user should input the number of credits necessary to graduate at their particular school in “Creds_To_Graduate”, and they should also input the StudentID in “Desired_StudentID” for the student they wish to get the credit count for.

- **calcGradCreds Function Example:**





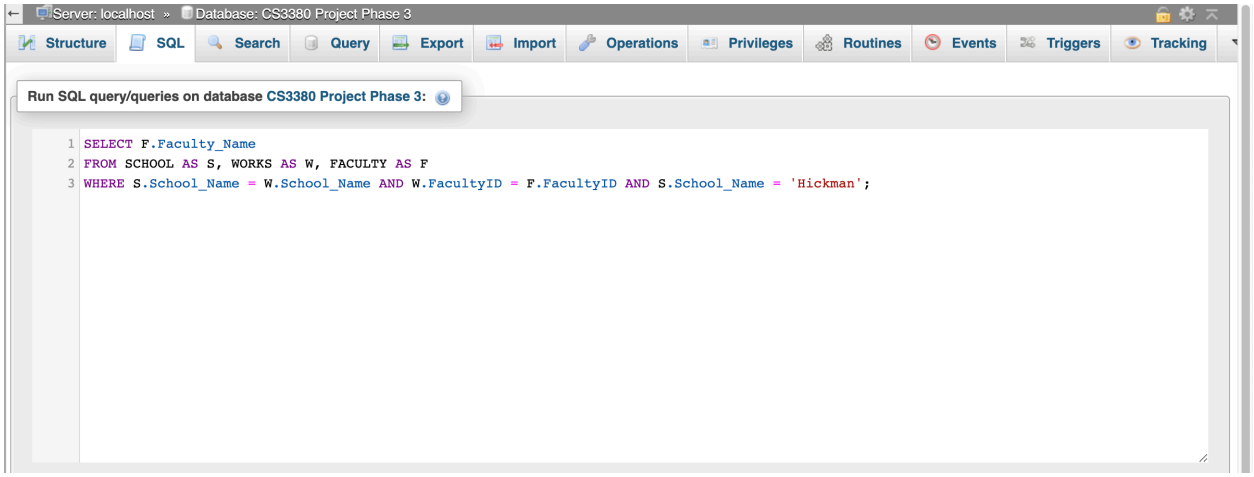
getSchoolFaculty Function:

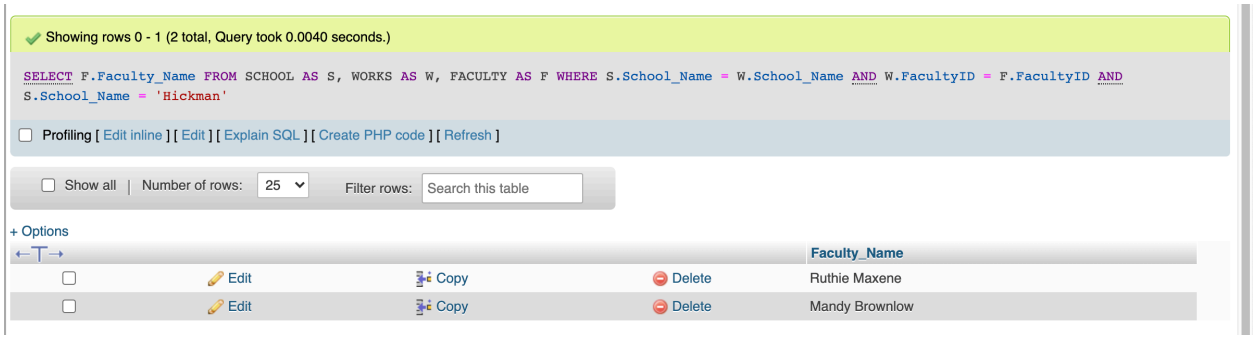
- **Description:** This function will get and return all the faculty members that work at a particular school.
- **Generalized Code:**

```
SELECT F.Faculty_Name
FROM SCHOOL AS S, WORKS AS W, FACULTY AS F
WHERE S.School_Name = W.School_Name AND W.FacultyID = F.FacultyID AND S.School_Name =
'School_Name';
```

- **Input and Usage:** For this function the user needs only to put in the “School_Name” for the school for which they wish to get the faculty members that work there. Like has been mentioned before, the school name must be entered exactly as it was first input into the databse in order to return correctly.

- **getSchoolFaculty Function Example:**





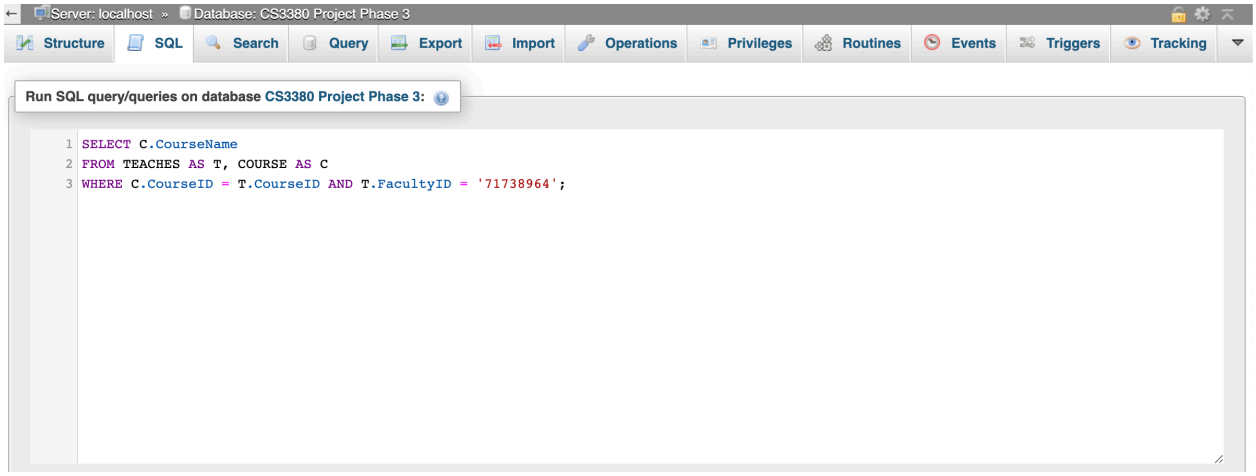
getFacultyCourses Function:

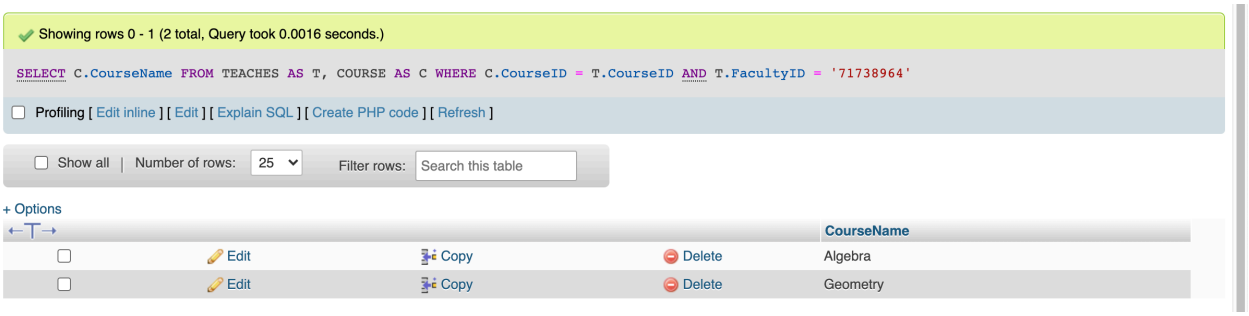
- **Description:** This function will return all the courses that a faculty member is currently teaching.
- **Generalized Code:**

```
SELECT C.CourseName
FROM TEACHES AS T, COURSE AS C
WHERE C.CourseID = T.CourseID AND T.FacultyID = 'Desired_FacultyID';
```

- **Input and Usage:** In this function the user must input the FacultyID in “Desired_FacultyID” for the faculty member of which they wish to get the courses for.

- **getFacultyCourses Function Example:**





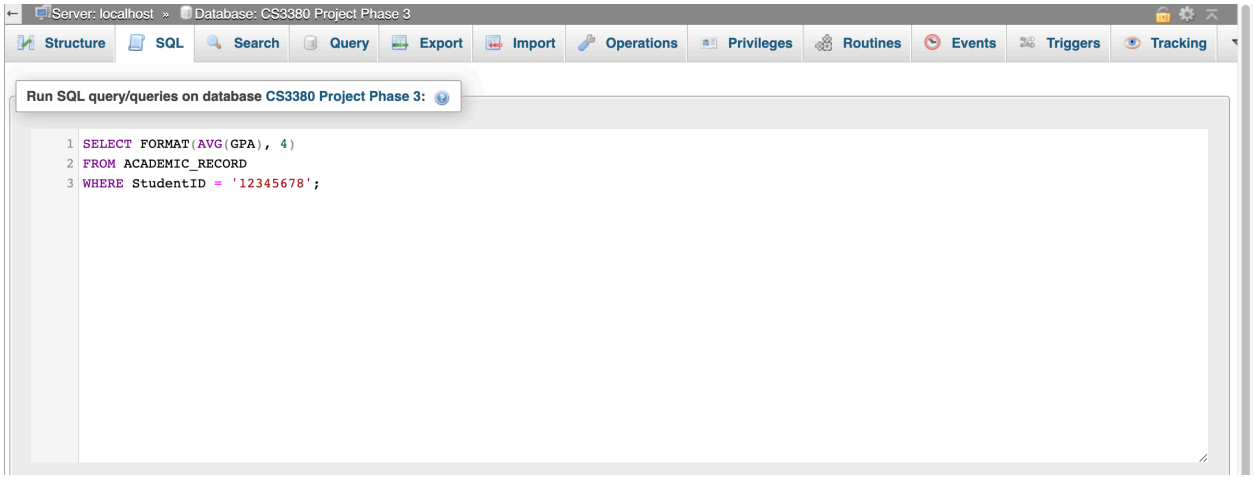
cumulativeGPA Function:

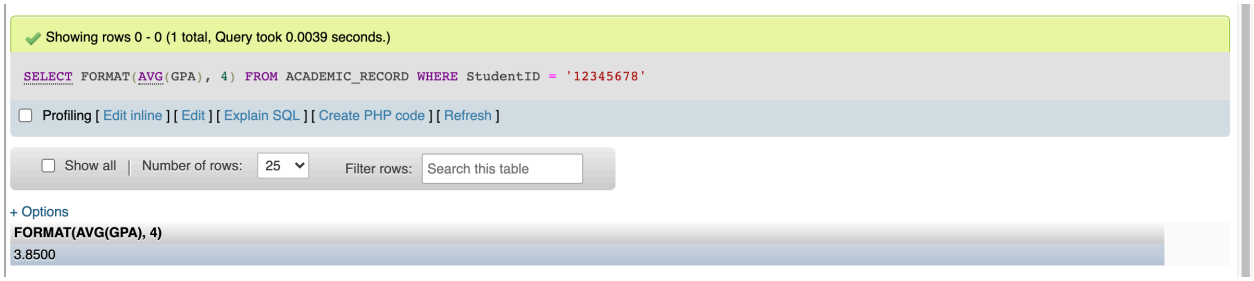
- **Description:** This function is used to get the cumulative GPA of a student by averaging their past semester GPAs.
- **Generalized Code:**

```
SELECT FORMAT(AVG(GPA), 4)
FROM ACADEMIC_RECORD
WHERE StudentID = 'Desired_StudentID';
```

- **Input and Usage:** For this function, the only thing the user must input is the StudentID in “Desired_StudentID” of the student they wish to get the cumulative GPA for. The GPA returned is automatically formatted to 4 decimal point precision in the function itself.

- **cumulativeGPA Function Example:**





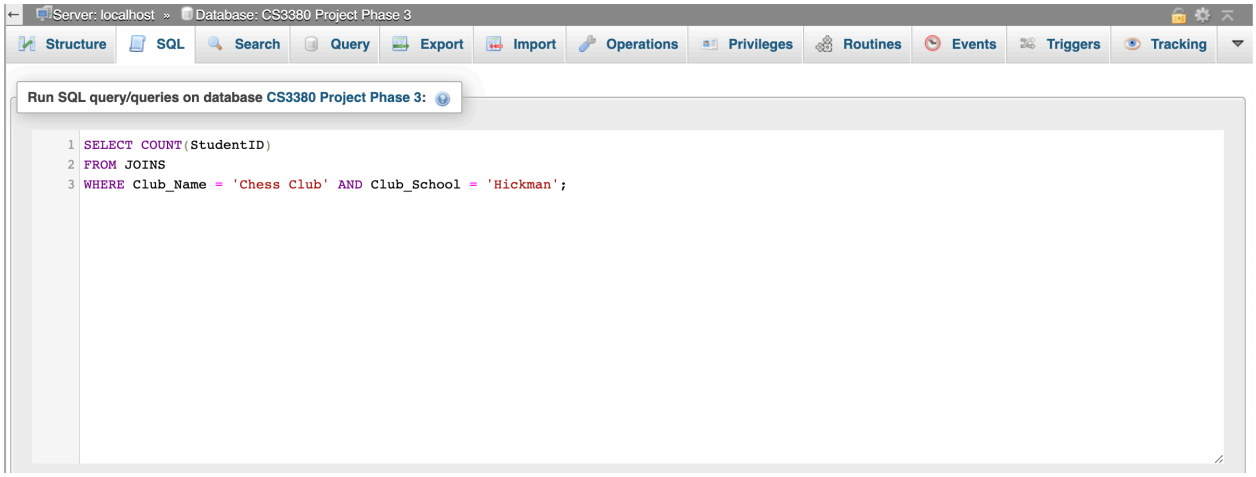
clubMemberCount Function:

- **Description:** This function is used to get the member count of a given club at a school.
- **Generalized Code:**

```
SELECT COUNT(StudentID)
FROM JOINS
WHERE Club_Name = 'Desired_Club_Name' AND Club_School = 'Desired_School_Name';
```

- **Input and Usage:** For this function, the user must input the name of the club in “Desired_Club_Name” and the school for which the club is hosted at “Desired_School_Name”. It will then return an integer of the count of members for the club.

- **clubMemberCount Function Example:**



getCoursePeriod Function:

- **Description:** This function is used to get the period of which a specific course at a specific school happens during.
- **Generalized Code:**

```
SELECT CP.ClassPeriod
FROM COURSE AS CO, CLASS_PERIOD AS CP
WHERE CO.CourseID = CP.CourseID AND CO.CourseName = 'Course_Name' AND CO.Course_School
= 'School_Name';
```

- **Input and Usage:** For this function, the user must input the name of the course “Course_Name” and the school for which the particular course is at “School_Name” in order to get the course period.

- **getCoursePeriod Function Example:**