Bio-Log Database Project

By Ary Hernandez, Jacquelyn Johnson, and Andrew Samuel

Spring 2020

Professor Yuan

**Table of Contents**

**Abstract**

**Mission Statement**

**Mission Objectives**

**Views**

**E/R Diagram**

**Normalization**

**Final E/R Diagram**

**Use Cases**

**Test & Outputs**

**Conclusion**

**References**

### Abstract

The proposed database will allow for a logical and concise storage of data collected from the Gen III MicroPlate Biolog lab being conducted at Lone Star College-Montgomery. This database application will allow students and professors the ability to query through large amounts of data collected, draw inferences from the data to extend hypothesis, and further the research in the field of microbiology pertaining to this specific lab. The platform is a MySQL database with the application GUI written in VB using Windows Forms App(.NET Framework). The result is a database which enables students and professors to enter, manage, and query data through a user-friendly interface.

### Mission Statement

The purpose of the Biolog database project is to maintain the data collected during the course of performing the steps associated with the Gen III Microplate Lab being conducted at Lone Star College-Montgomery under the direction of Dr. Julie Harless. This database will allow for data to be organized in such a way that hypothesis can be formulated and tested . The students and professors will be able to query the data in such a way to allow for new labs to be developed and further their research into the electrical output of microorganisms.

### Mission Objectives

To maintain (enter, update, and delete) data on State

To maintain (enter, update, and delete) data on University

To maintain (enter, update, and delete) data on Campus

To maintain (enter, update, and delete) data on Professor

To maintain (enter, update, and delete) data on Class

To maintain (enter, update, and delete) data on Semester

To maintain (enter, update, and delete) data on Location

To maintain (enter, update, and delete) data on Student

To maintain (enter and update) data on Data

To perform searches on State

To perform searches on University

To perform searches on Campus

To perform searches on Class

To perform searches on Semester

To perform searches on Location

To perform searches on Student

To perform searches on Data

To track status of Data

To report on State

To report on University

To report on Campus

To report on Professor

To report on Class

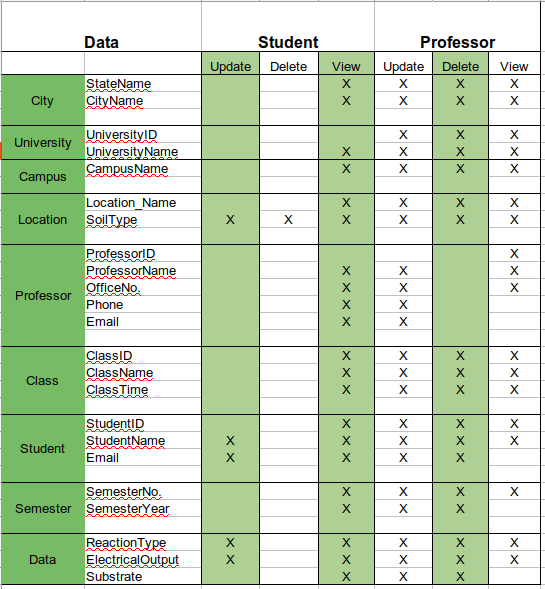
To report on Semester

To report on Location

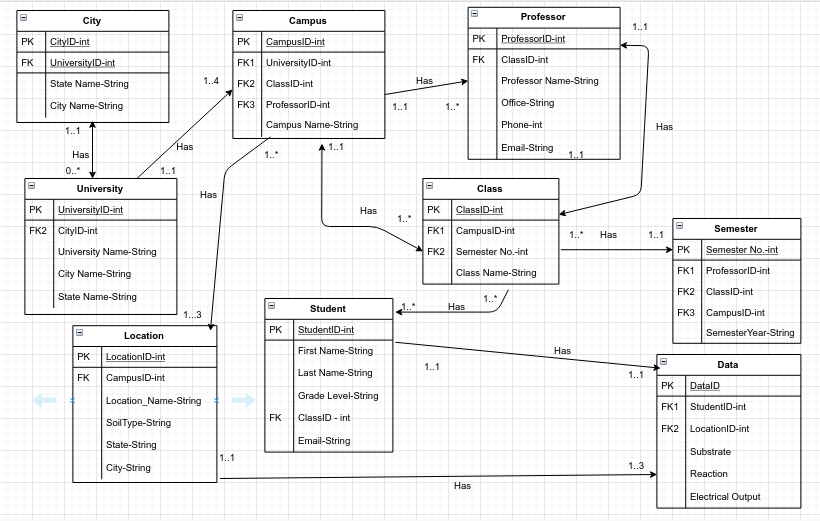
To report on Student

To report on Data

### Major User Views



### E/R Diagram



### Table Explanations:

1. City

The purpose of the ‘City’ table is to hold the city name and state name where the university is located. Since there can be more than one university in a city, this table is useful when querying information. This table allows for the expansion of the use of this application/database as this lab is available to be performed at any university interested in the electrical output of bacteria in different soil types. The primary key for this table is CityID and the foreign key for this table is UniversityID.

2. University

The purpose of the ‘University’ table is to hold the information about the university at which the lab is being conducted. This table will hold the university name and has the foreign key CityID. The primary key for the table is UniversityID.

3. Campus

The purpose of the ‘Campus’ table is to hold information about the exact campus at which the lab is being performed. The logic behind this is as follows: University of Houston has campuses in a variety of places, Clear Lake, Downtown, Victoria, etc. There are many universities that also have multiple campuses, like Lone Star College System, for which this database is being designed. By implementing a ‘Campus’ table, the data collected at the different campuses can be kept distinct. The primary key for this table is CampusID and the foreign keys are UniversityID, ClassID, and ProfessorID.

4. Location

The purpose of the ‘Location’ table is to hold the data for the different, distinct locations that will be studied at each campus. The purpose of the lab itself is to “see” the reactions to the substrate that the bacteria have where the bacteria were collected from different soil conditions. For example, samples could be collected from pond mud, dry dirt, and loamy soil respectively. As each of the locations would be unique, there is a need for a table to hold the data collected about each one. The primary key for this table is LocationID and the foreign key is CampusID.

5. Student

The purpose of the ‘Student’ table is to hold identifying information about the student performing the lab and collecting the data. The information held in this table is the name of the student, what year they are in concerning their studies, their email, and the password that they will use to log in to the user interface and document the progress of the lab they are working on. As there can be many students who complete this lab across all of the possible campuses, this table is necessary. The primary key for this table is StudentID and the foreign key for this table is ClassID.

6. Professor

The purpose of the ‘Professor’ table is to hold identifying information about the professor at the campus where the lab is being conducted. This table will contain the following information about the professor: name, office, email, and phone as query-able data. It is possible that there can be more than one professor at a campus that is assisting students in the collection of data and completion of the lab. The primary key of this table is ProfessorID and the foreign key is ClassID.

7. Class

The purpose of the ‘Class’ table is to hold information about the classes to which the students belong. This is necessary as there can be multiple sections of a class offered at a time in which there are students in different sections that are performing the same lab. The class name will also contain the section number within it so as to differentiate between the unique classes. The primary key for this table is ClassID and the foreign keys are CampusID and SemesterNo.

8. Semester

The purpose of the ‘Semester’ table is to hold identifying information concerning the semester during which the lab was performed: fall, spring, and summer. It will also contain the year during which the semester occurs to further differentiate the time and aid in future comparative queries of the data. The primary key for this table is SemesterNo and the foreign keys for this table are ProfessorID, ClassID, and CampusID.

9. Data

The purpose of the ‘Data’ table is to hold all of the data that is collected over the course of the lab. This table will be modified the most for each lab, mainly through updating reactions and the resulting electrical output for the substrate being tested based on the location that the sample was collected from. The primary key for this table is DataID and the foreign keys are StudentID and LocationID.

### Use Cases

**Entity: City**

1. Insert New City

Actor: Professor

Steps:

* User clicks “New City” button
* User sees a form appear to collect information
* User inputs information for the State name and the city name
* User clicks the “Continue” button

MySQL statement:

INSERT INTO City (stateName, cityName) VALUES (stateName, cityName);

Explanation: This use case allows the Professor to add a new city where a new campus is located that will be collecting data for this lab.

2. Delete City

Actor: Professor

Steps:

* User clicks “Delete City” button
* User sees a window displaying all cities in the database
* User enters the cityID number in a text field that they wish to delete
* User clicks “Delete” button
* User sees a dialog box warning that this action cannot be undone
* User clicks “Confirm Deletion” button on the dialog box

MySQL statement:

DELETE FROM City WHERE cityID = cityID;

Explanation: This use case will allow for the deletion of a city

3. View Cities

Actor: Professor/ Student

Steps:

* User clicks “Show Cities” button
* User sees a window that displays the contents of the City table

MySQL statement:

SELECT (cityName, stateName) FROM City;

Explanation: This use case allows the system users to view the cities in the table

4. Update City

Actor: Professor

Steps:

* User clicks “Show Cities” button
* User sees a window that displays the contents of the City table
* User clicks “Update City” button
* User inputs the cityID of the city to be updated
* User is prompted for the new cityName and stateName
* User inputs newCityName and newStateName into labeled fields
* User clicks “Update Record”

MySQL statement:

UPDATE City SET cityName = newCityName WHERE cityName = cityName\_original

UPDATE City SET stateName=newStateName WHERE stateName=stateName\_original

Explanation: This use case allows a professor to correct the city/state of a record

5. Count Cities

Actor: Professor

Steps:

* User clicks “Count Cities” button
* User sees a window that displays the count of cities by state

MySQL statement:

SELECT COUNT(cityID), stateName

FROM City

GROUP BY stateName

Explanation: This use case displays the number of cities by state

**Entity: University**

6. Insert New University

Actor: Professor

Steps:

* User clicks “New University” button
* User sees a form appear to collect information
* User inputs information for the university name
* User clicks the “Continue” button

MySQL statement:

INSERT INTO University (universityName) VALUES (universityName);

Explanation: This use case allows a professor to add a new university record to the University table

7. Delete University

Actor: Professor

Steps:

* User clicks “Delete University” button
* User sees a list of universities
* User inputs the universityID of the university to delete
* User clicks “Delete” button
* User sees a message dialog box informing that this action cannot be undone
* User clicks “Confirm Deletion” button

MySQL statement:

DELETE FROM University WHERE universityID=universityID;

Explanation: This use case allows a professor to delete a university from the University table

8. View Universities

Actor: Professor/Student

Steps:

* User clicks “Show Universities” button
* User sees window showing all universities in the University table

MySQL statement:

SELECT \* FROM University;

9. Count Universities

Actor: Professor/Student

Steps:

* User clicks “Count Universities” button
* User is displayed a window with number of universities by state

MySQL statement:

SELECT COUNT(uniID) stateName

FROM UNIVERSITY

GROUP BY stateName

Explanation: User can see the number of universities by state.

10. Update University

Actor: Professor

Steps:

* User clicks “Show Universities” button
* User sees window showing all universities in the University table
* User clicks “Update University” button
* User inputs universityID of university record to update
* User is prompted for new university name
* User clicks “Update Record” button

MySQL statement:

UPDATE University SET universityName=newUnivName WHERE universityName=universityName\_original;

Explanation: This use case allows the professor to update the name of a university in the University table

**Entity: Campus**

11. Insert New Campus

Actor: Professor

Steps:

* User clicks “Add Campus” button
* User selects the city where the new campus is located
* User selects the university the campus belongs to; universityID is captured
* User inputs new campus name
* User clicks “Continue” button

MySQL statement:

INSERT INTO Campus (campusName, universityID) VALUES (campusName, universityID);

Explanation: This use case allows a campus to be added to the Campus table and foreign key to be linked at creation of record

12. Delete Campus

Actor: Professor

Steps:

* User clicks “Delete Campus” button
* User sees a window displaying the contents of the Campus table
* User inputs the campusID of the campus to be deleted
* User clicks “Delete” button
* User sees a message dialog box informing the user that this action is not reversible
* User clicks “Confirm Deletion” button

MySQL statement:

DELETE FROM Campus WHERE campusID=campusID;

Explanation: This use case allows a professor to delete a campus from the Campus table

13. View Campuses

Actor: Professor/Student

Steps:

* User clicks “View Campuses” button
* User sees a window displaying the contents of the Campus table

MySQL statement:

SELECT \* FROM Campus;

Explanation: This use case will display the contents of the Campus table for all users

14. Count Campuses

Actor: Professor/Student

Steps:

* User selects “Count Campuses” button
* User will be presented a window with number of campuses by university

MySQL statement:

SELECT COUNT(campusID), universityName

FROM Campus

GROUP BY universityName

Explanation: With this, the user can see the number of campuses by university system. Some university systems have multiple campuses. Ex. (UH system)

15. Update Campus

Actor: Professor

Steps:

* User clicks “View Campuses” button
* User sees a window displaying the contents of the Campus table
* User clicks “Update Campus” button
* User is prompted for the campusID of the campus to update
* User inputs the new campus name
* User clicks “Update Record” button

MySQL statement:

UPDATE Campus SET campusName=newCampusName WHERE campusName=campusName\_original;

Explanation: This use case updates the name of the campus

**Entity: Location**

16. Insert New Location

Actor: Professor

Steps:

* User clicks “Add New Location” button
* User sees a form pop up to collect information
* User selects the campus on which the location is found; campusID captured
* User inputs the name of the location and soil type found there
* User clicks “Continue” button

MySQL statement:

INSERT INTO Location (locationName, soilType, campusID) VALUES (locationName, soilType, campusID);

Explanation: This use case adds a new location to the Location table and foreign key of campusID to be linked at creation of record

17. Delete a Location

Actor: Professor

Steps:

* User clicks “Delete Location” button
* User inputs locationID of location to delete
* User clicks “Delete” button
* User sees a message dialog box informing them that the action cannot be reversed
* User clicks “Confirm Deletion” button

MySQL statement:

DELETE FROM Location WHERE locationID=locationID;

Explanation: This use case allows a location to be deleted

18. View Locations

Actor: Professor/Student

Steps:

* User clicks “View Locations” button
* User sees a window displaying the contents of the Location table

MySQL statement:

SELECT \* FROM Location;

Explanation: This use case allows a user to view all the locations in the Location table

19. Update Location Name

Actor: Professor

Steps:

* User clicks “View Locations” button
* User sees a window displaying the contents of the Location table
* User clicks “Update Location” button
* User is prompted for the locationID of the location record to be updated
* User inputs the locationID
* User inputs new location name
* User clicks “Update Record” button

MySQL statement:

UPDATE Location SET locationName=newLocationName WHERE locationName=locationName\_original;

Explanation: This use case allows a location name to be updated by the professor

20. Update Location Soil Type

Actor: Professor/Student

Steps:

* User clicks “View Locations” button
* User sees a window displaying the contents of the Location table
* User clicks “Update Location Soil Type” button
* User is prompted for the locationID of the location record to be updated
* User inputs the locationID
* User inputs new soil type
* User clicks “Update Record” button

MySQL statement:

UPDATE Location SET soilType=newSoilType WHERE soilType=soilType\_original;

Explanation: This use case allows a soil type to be updated at a location by the professor or the student

21. Delete a Location Soil Type

Actor: Professor

Steps:

* User clicks “Delete Soil Type” button
* User inputs locationID of location where soil type needs to be deleted
* User sees soil type listed for that location
* User inputs the name of the soil type
* User clicks “Delete” button
* User sees a message dialog box informing them that the action cannot be reversed
* User clicks “Confirm Deletion” button

MySQL statement:

UPDATE Location SET soilType=NULL WHERE soilType=soilType\_original;

Explanation: This use case allows a soil type to appear deleted to the user and inserts a NULL value in its place in the Location table

22. Count Soil Type

Actor: Professor

Steps:

* User clicks on “Count Soil Types” button
* User is displayed the count of soil types by location

mySQL statement:

SELECT COUNT(locationID), soil\_type

FROM Location

GROUP BY soil\_type

Explanation: The user can see the types of soil found by location.

**Entity: Professor**

23. Insert New Professor

Actor: Professor

Steps:

* User clicks “Add Professor” button
* User verifies their own credentials to maintain integrity of the database
* User sees a form pop up to collect information
* User inputs professor name, office, phone, and email
* User clicks “Continue”

MySQL statement:

INSERT INTO Professor (professorName, office, phoneNumber, email) VALUES (professorName, office, phoneNumber, email);

Explanation: This use case allows for the creation of a new record into the Professor table

24. Delete Professor

Actor: Professor

Steps:

* User clicks “Delete Professor”
* User verifies their own credentials to maintain integrity of the database
* User sees a window displaying the current records in the Professor table
* User is prompted for the professorID to delete
* User inputs the professorID
* User clicks “Delete”
* User sees a message dialog box informing the user that this action is irreversible
* User clicks “Confirm Deletion” button

MySQL statement:

DELETE FROM Professor WHERE professorID=professorID;

Explanation: This use case allows a record to be deleted from the professor table

25. View Professor Information

Actor: Professor/Student

Steps:

* User clicks “View Professors” button
* User sees a window displaying all the professors in the Professor table

MySQL statement:

SELECT \* FROM Professor;

Explanation: This use case allows users to view the records in the Professor table

26. Update Professor Information

Actor: Professor

Steps:

* User clicks “View Professors” button
* User sees a window displaying all the professors in the Professor table
* User clicks “Update Professor” button
* User verifies their own credentials to maintain integrity of the database
* User is prompted for the professorID of the record to update
* User inputs the professorID of the record to update
* User selects the information to be updated
* User inputs the new values for the information that needs to be updated
* User clicks “Update Record”

MySQL statement:

UPDATE Professor SET professorName=newProfessorName WHERE professorName=professorName\_original;

UPDATE Professor SET office=newOffice WHERE office=office\_original;

UPDATE Professor SET phoneNumber=newPhone WHERE phoneNumber=phoneNumber\_original;

UPDATE Professor SET email=newEmail WHERE email=email\_original;

Explanation: This use case allows for a professor record to be updated. The MySQL statement that will be executed depends wholly on the selected information and will only update the information that is selected to be updated. This will be verified and handled programmatically on the GUI side.

27. Count Professor

Actor: Professor/Student

Steps:

* User clicks “Count Professor” button
* User is displayed number of professor teaching course ID 1100

MySQL statement:

SELECT count(professor\_ID) AS "Professors Teaching this Class"

FROM Professor WHERE (class\_ID = 1100);

Explanation: Shows number of professor teaching a specific class

**Entity: Class**

28. Insert New Class

Actor: Professor

Steps:

* User clicks “Add New Class” button
* User sees a form pop up to collect information for the new record
* User selects the campus and semester from cascading menus; campusID and semesterID are captured
* User inputs class name and class time
* User clicks “Add”

MySQL statement:

INSERT INTO Class (className, classTime, campusID, semesterID) VALUES (className, classTime, campusID, semesterID);

Explanation: This use case adds a new record to the Class table and links the foreign keys for campusID and semesterID at creation.

29. Delete Class

Actor: Professor

Steps:

* User clicks “Delete Class”
* User verifies their credentials to maintain database integrity
* User sees window displaying contents of Class table
* User prompted for classID of class to delete
* User inputs classID
* User clicks “Delete”
* User sees pop up message informing the action is irreversible
* User clicks “Confirm Deletion”

MySQL statement:

DELETE FROM Class WHERE classID=classID;

Explanation: Allows for deletion of a class record from the Class table

30. View Classes

Actor: Professor/Student

Steps:

* User clicks “View Classes”
* User sees window displaying all records in Class table

MySQL statement:

SELECT \* FROM Class;

31. Update Class

Actor: Professor

Steps:

* User clicks “View Classes”
* User sees window displaying all records in Class table
* User clicks “Update Class”
* User verifies credentials
* User prompted for classID
* User inputs classID
* User inputs new class name and class time
* User clicks “Update Record”

MySQL statement:

UPDATE Class SET className=newClassName WHERE className=className\_original;

UPDATE Class SET classTime=newClassTime WHERE classTime=classTime\_original;

Explanation: Allows for the name and time of a class to be updated

32. Count Classes

Actor: Professor/Student

Steps:

* User clicks on “Count Classes” button
* User is displayed count of classes by semester

MySQL statement:

SELECT COUNT(classID), semesterNo

FROM Class

GROUP BY semesterNo

Explanation: The user can see the number of classes by each campus

**Entity: Student**

33. Insert New Student

Actor: Professor

Steps:

* User clicks “Add New Student”
* User verifies credentials
* User sees a form pop up to collect information for record
* User inputs student name (first and last), grade level, and email
* User selects the class to which the new student belongs; classID captured
* User clicks “Continue”

MySQL statement:

INSERT INTO Student (firstName, lastName, gradeLevel, email) VALUES (firstName, lastName, gradeLevel, email);

Explanation: creates a new record in the Student table and links classID as foreign key to the record

34. Delete Student

Actor: Professor

Steps:

* User clicks “Delete Student”
* User verifies credentials
* User sees all student records from Student table
* User prompted for studentID to delete
* User inputs studentID
* User clicks “Delete”
* User sees message stating the action is irreversible
* User clicks “Confirm Deletion”

MySQL statement:

DELETE FROM Student WHERE studentID=studentID;

Explanation: Allows a student record to be deleted

35. View Students

Actor: Professor/Student

Steps:

* User clicks “View Students”
* User sees window displaying contents of Student table

MySQL statement:

SELECT \* FROM Student;

Explanation: Displays contents of Student table

36. Update Student

Actor: Professor/Student

Steps:

* User clicks “View Students”
* User sees window displaying contents of Student table
* User selects record to update
* User verifies credentials; if student actor, must verify that record is their own
* User selects information to update
* User inputs new information for selected fields
* User clicks “Update Record”

MySQL statement:

UPDATE Student SET firstName=newFirstName WHERE firstName=firstName\_origin;

UPDATE Student SET lastName=newLastName WHERE lastName=lastName\_origin;

UPDATE Student SET gradeLevel=newGrLevel WHERE gradeLevel=grLevel\_origin;

UPDATE Student SET email=newEmail WHERE email=email\_origin;

Explanation: Allows a student/professor to update information in a student record

37. Count Students

Actor: Professor/Student

Steps:

* User clicks “Count Students”
* User sees window displaying number of student’s grade level

MySQL statement:

SELECT COUNT(studentID), grade\_level

FROM Student

GROUP BY grade\_level

Explanation: The user will be able to see the number of freshmen, sophomores, juniors and seniors per campus

**Entity: Semester**

38. Insert New Semester

Actor: Professor

Steps:

* User clicks “Add New Semester”
* User selects professor, campus, and class from menus; professorID, campusID, classID captured
* User inputs semester year and semester season
* User clicks “Continue”

MySQL statement:

INSERT INTO Semester (semesterYear, semesterSeason, professorID, classID, campusID) VALUES (semesterYear, semesterSeason, professorID, classID, campusID);

Explanation: creates new record in Semester table and links foreign keys for professorID, classID, campusID upon creation

39. Delete Semester

Actor: Professor

Steps:

* User clicks “Delete Semester”
* User verifies credentials
* User sees contents of Semester table
* User inputs semesterNO of semester to delete
* User clicks “Delete”
* User informed that action is irreversible
* User clicks “Confirm Deletion”

MySQL statement:

DELETE FROM Semester WHERE semesterNO=semesterNO;

Explanation: Deletes record from Semester table

40. View Semesters

Actor: Professor

Steps:

* User clicks “View Semesters”
* User sees window displaying contents of Semester table

MySQL statement:

SELECT \* FROM Semester;

Explanation: Displays contents of Semester table

41. Update Semester

Actor: Professor

Steps:

* User clicks “View Semesters”
* User sees window displaying contents of Semester table
* User clicks “Update Semester”
* User verifies credentials
* User inputs new semester year and new semester season
* User clicks “Update Record”

MySQL statement:

UPDATE Semester SET semesterYear=newSemYear WHERE semesterYear=semYear\_origin;

UPDATE Semester SET semesterSeason=newSeason WHERE semesterSeason=semesterSeason\_origin;

Explanation: Updates information in semester record

42. Count Semester

Actor: Professor/Student

Steps:

* User clicks “Count Semesters”
* User is displayed number of semesters by semester season (Spring, Fall or Summer)

MySQL statement:

SELECT COUNT(semesterNo), semesterSeason

FROM Semester

GROUP BY semesterSeason

Explanation: Shows number of semesters by season.

**Entity: Data**

43. Insert New Lab Data

Actor: Student

Steps:

* User clicks “New Lab Data”
* User inputs their studentID
* User selects semester and location from menu; semesterNO and locationID captured
* User sees form pop up for information collection
* User inputs reaction type and electrical output for selected substrate
* User clicks “Continue”

MySQL statement:

INSERT INTO Data (reactionType, electricalOutput, substrate, studentID, semesterNO, locationID) VALUES (reactionType, electricalOutput, substrate, studentID, semesterNO, locationID);

Explanation: Adds a new record to Data that holds the lab data being collected by that student and links the foreign keys studentID, semesterNO, and locationID at creation.

44. Delete Data

Actor: Professor

Steps:

* User clicks “Delete Lab Data”
* User verifies credentials
* User sees window displaying contents from Data table
* User inputs dataID of record to delete
* User clicks “Delete”
* User informed action is irreversible
* User clicks “Confirm Deletion”

MySQL statement:

DELETE FROM Data WHERE dataID=dataID;

Explanation: Deletes record from table

45. View Lab Data

Actor: Professor/Student

Steps:

* User clicks “View Lab Data”
* User selects studentID
* User sees window displaying contents of Data table for selected studentID

MySQL statement:

SELECT \* FROM Data WHERE studentID=studentID;

Explanation: Displays lab data of selected studentID from Data table

46. Update Lab Data

Actor: Professor/ Student

Steps:

* User clicks “View Lab Data”
* User selects studentID
* User sees window displaying contents of Data table for selected studentID
* User clicks “Update Lab Data”
* User selects dataID for record to update
* User inputs data for selected substrate
* User clicks “Update Record”

MySQL statement:

UPDATE Data SET reactionType=newReaction WHERE (reactionType=reactioType\_origin, substrate=substrate\_origin);

UPDATE Date SET electricalOutput=newElec WHERE (electricalOutput=electricalOutput\_origin, substrate=substrate\_origin);

Explanation: allows a user to update a record

47. Count Data

Actor: Professor/Student

Steps:

* User clicks “Count Data” button
* User is displayed number of data organized by each reaction type

MySQL statement:

SELECT COUNT(dataID), reactionType

FROM Data

GROUP BY reactionType

Explanation: The user can the see the number of reactions by reaction type.

48. Average Electrical Output

Actor: Professor/Student

Steps:

* User clicks “Average Electrical Output”
* User is displayed lists of Electrical outputs and their average

MySQL statement:

SELECT AVG (electricalOutput)

FROM Data

Explanation: Display the averages of recorded electrical outputs

Join City & University

SELECT city\_name.City, state\_name.City, university\_name.University, campus\_name.Campus

FROM City JOIN University ON City.city\_ID = University.city\_ID

JOIN Campus ON University.university\_ID = Campus.university\_ID;

Join Campus & Semester

SELECT campus\_name.Campus, term\_name.Semester, term\_year.Semester, class\_name.Class

FROM Campus JOIN Semester ON Campus.campus\_ID = Semester.campus\_ID

JOIN Class ON Semester.term\_ID = Class.term\_ID;

Join Semester and Class

SELECT term\_name.Semester, term\_year.Semester, class\_ID.Class, last\_name.Professor, class\_name.Class, class\_room.Class

FROM Semester JOIN Class ON Semester.term\_ID = Class.term\_ID

JOIN Professors ON Class.class\_ID = Professor.class\_ID;

Join Campus & Location

SELECT campus\_name.Campus, location\_name.Location, data\_ID.Data\_Yield, substrate.Data\_Yield, reaction.Data\_Yield, electrical\_output.Data\_Yield

FROM Campus JOIN Location ON Campus.campus\_ID = Location.campus\_ID

JOIN Data\_Yield ON Location.location\_ID = Data\_Yield.location\_ID;

Join Class & Student

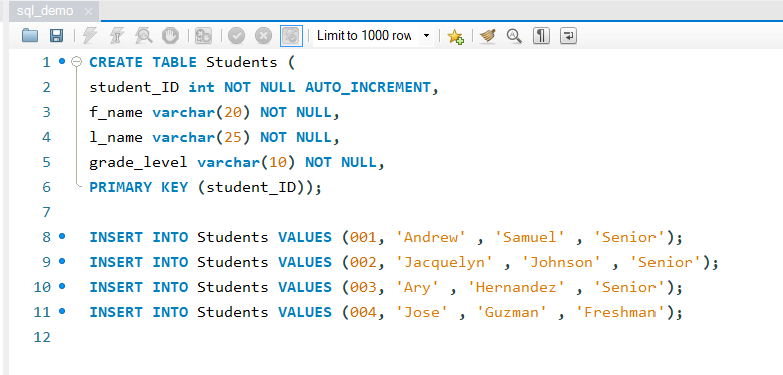
SELECT class\_name.Class, first\_name.Student, last\_name.Student, data\_ID.Data\_Yield, substrate.Data\_Yield, reaction.Data\_Yield, electrical\_output.Data\_Yield

FROM Class JOIN Student ON Class.class\_ID = Student.class\_ID

JOIN Data\_Yield ON Student.student\_ID = Data\_Yield.student\_ID;

### Database Prototype

For the implementation of this database, the tools that will be used are as follows: MySQL, MySQL Workbench, Ubuntu terminal, Visual Studio, and .NET framework web application. The database development will take place in the MySQL Workbench environment, allowing for the developer see precisely what is taking place during the programming of the database. Naturally, the database language that will be used is MySQL as it easily translates within the Workbench environment and the Visual Studio environment. Visual Studio will be used to develop the GUI of the application that the end user will be using to interact with the database. An example of the initial population of this database will look like this:

 Once the database has been tested thoroughly for bugs, it will be populated with the data collected from actual lab data last semester by a student at Lone Star College-Montgomery and a partial-live test will be conducted with the GUI to verify that the database and GUI work as expected by the client, Dr. Harless.

### Project Timetable

**Task 1: Finalize needed tables and attributes**

**Members responsible for task: Ary Hernandez, Andrew Samuel, Jacquelyn Johnson**

As a democratic team, all members will review the tables and attributes found within the database to make certain that there aren't any entities or attributes being overlooked.

**Task 2: Write code using MySQL for the database**

**Members responsible for task: Andrew Samuel (lead), Ary Hernandez, Jacquelyn Johnson**

The team will meet each week, at least once weekly, to write the code for the database from March 17, 2020 through April 7, 2020. For this task, Andrew Samuel will take lead and assign tasks to the other members of the team until the database is fully coded.

**Task 3: Implement database within an application**

**Members responsible for task: Jacquelyn Johnson (lead), Ary Hernandez, Andrew Samuel**

The team will work together for the design and implementation of the database within an application. As this project is beign designed for delivery to another university that uses Microsoft exclusively, this application will be written in VB for ease of installation and use on the machines that are currently being used by the university. For this task, Jacquelyn Johnson will take the lead, assign tasks to the other members of the team, and call meetings as needed until the database is fully implemented. This task should be completed by April 21, 2020, leaving a week to make any necessary changes to visual presentation and application logic answering the question: does the layout make sense to an end user.

