Table of Contents

[Introduction 2](#_Toc39497585)

[Document Layout 2](#_Toc39497586)

[MySQL database tutorial 2](#_Toc39497587)

[Setting up a connection with MySQL 3](#_Toc39497588)

[Parameters for MySQL database connection: 3](#_Toc39497589)

[Methodology of MySQL database 3](#_Toc39497590)

[Overview of entire workflow (visual) 4](#_Toc39497591)

[Computer Setup to allow for the Python import process 4](#_Toc39497592)

[Continuous Water Quality Sensor Types 5](#_Toc39497593)

[Discrete Water Quality Sensor Types 7](#_Toc39497594)

[Service Log Workflow (mandatory for every water quality service trip) 9](#_Toc39497595)

[Scenario: I did the Python import process, and I found a WARNING message in my import process output report. What do I do? 14](#_Toc39497596)

[Scenario: I just imported the continuous water quality data using the python import process and there are YSI measurements from the water quality datasheet that I need to manually enter into MySQL. How do I do this? 15](#_Toc39497597)

[Scenario: I was just forwarded Lakewatch results. What do I do with them? 16](#_Toc39497598)

[Scenario: The LCR project bought several new Star-Oddi/Diver sensor devices, how do I account for these new sensors in MySQL? 17](#_Toc39497599)

[Scenario: The LCR project has added new water quality sites, where do I specify this in MySQL? 18](#_Toc39497600)

[Scenario: I replaced a sensor in the field, how do I tell MySQL about this? 18](#_Toc39497601)

[Scenario: I accidently deleted a whole table in MySQL, what do I do now? 19](#_Toc39497602)

[Recommendations: MySQL tables to become familiar with 20](#_Toc39497603)

[Table Definitions (listed in the order in the MySQL Workbench) 21](#_Toc39497604)

[Table: lcroyster\_buoyobservation 21](#_Toc39497605)

[Table: lcroyster\_location 23](#_Toc39497606)

[Table: lcroyster\_project 24](#_Toc39497607)

[Table: lcroyster\_sensor 26](#_Toc39497608)

[Table: lcroyster\_sensordeploy 28](#_Toc39497609)

[Table: lcroyster\_sensorservice 29](#_Toc39497610)

[Table: lcroyster\_sensortype 31](#_Toc39497611)

[Table: lcroyster\_waterobservation 32](#_Toc39497612)

[Resources at UF: 34](#_Toc39497613)

**LCR MySQL Workflow Theory**

# Introduction

This document is to provide insight into how the MySQL database is utilized and accessed in the Lone Cabbage Reef (LCR) restoration project. The MySQL database is a custom database specifically for water quality observations, both continuous and discrete. Development of the database was led by the UF Marston Library Academic Research Computing group. The database was motivated by the need to store data from multiple water quality variables collected on an hourly basis from an array of water quality sensors. These sensors were deployed at specific water quality stations that were identified locations to track water quality responses to the LCR restoration project and other factors such as changes in Suwannee River discharge. In this instance the location of the sensor was critically important, not the individual sensor (identified by serial number) itself, because we recognized that the sensor at a given location may change over time due to sensor malfunction or loss. This created a need in the database where all data collected from individual sensors had to be linked to the correct water quality station even as individual sensors at a given water quality station may have changed over time. Tables in the databased are linked together by columns, by location (*location\_id*) and sensor serial number (*sensor\_id*). This is not always the case, and those links and connections will be described later in this document. This structure links tables together such that the tables inform each other and help to create data integrity standards within the database, and to track the movement of specific sensors in specific locations. This MySQL database will by design store all water quality observations regardless of their initial reliability. During the initial water quality import process (through Python coding), checks such as duplicate observation are flagged and are described in the import process output report. Additional QA/QC steps are performed after the fact in R.

## Document Layout

This document will describe various MySQL scenarios, specific to the LCR project, and ways to fix or go about completing a process. This document will then describe each table in detail.

# MySQL database tutorial

It is imperative that time be taken to understand general MySQL functions, views, and codes to produce views. Additional resources about MySQL are found here:

<https://www.mysqltutorial.org/>

# Setting up a connection with MySQL

Download the MySQL Workbench version required for your machine:

<https://www.mysql.com/products/workbench/>

Set up a connection using the sets below and add the parameters to the applicable fields.

<https://dev.mysql.com/doc/workbench/en/wb-mysql-connections-new.html>

## Parameters for MySQL database connection:

Please review the MySQL Parameters document.

# Methodology of MySQL database

The current database structure is based on data tables linked to each other through the spatial location and sensor serial number to track an individual sensor through space and time. Sensors are often replaced in the field (due to damage, biofouling, or internal errors) and it is important to know which sensors are in which locations to maintain a continuous data stream from one location in time. The MySQL database was designed specifically to store water quality data and keep records of what sensors are active in the water quality sites. The methodology includes accounting for each individual sensor, each water quality location, and all associated information needed to know the history of a water quality site and a specific sensor.

# Overview of entire workflow (visual)

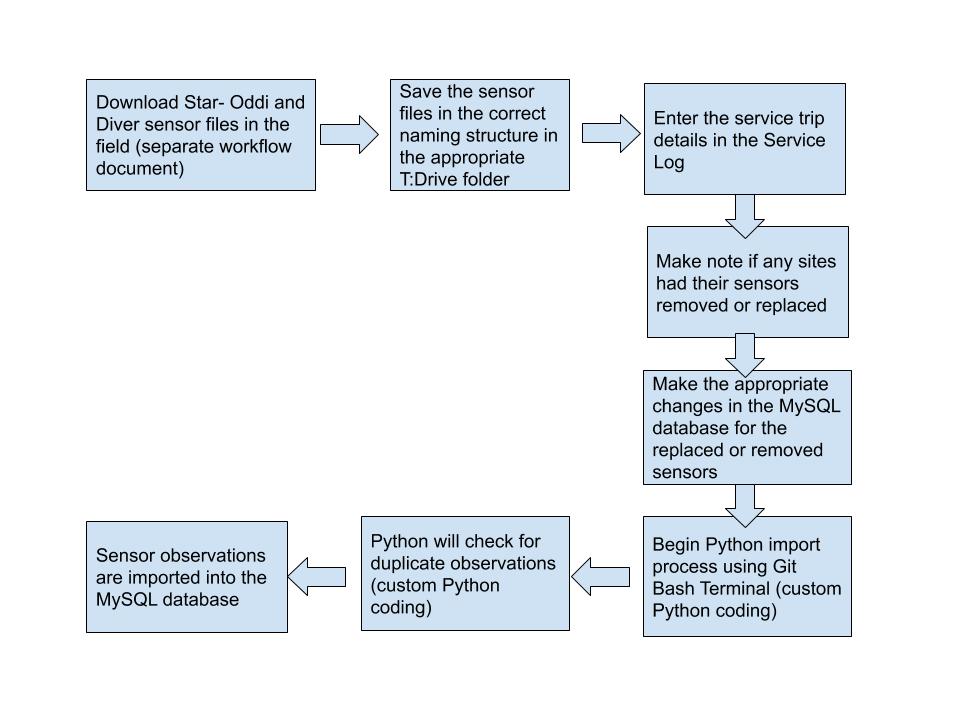
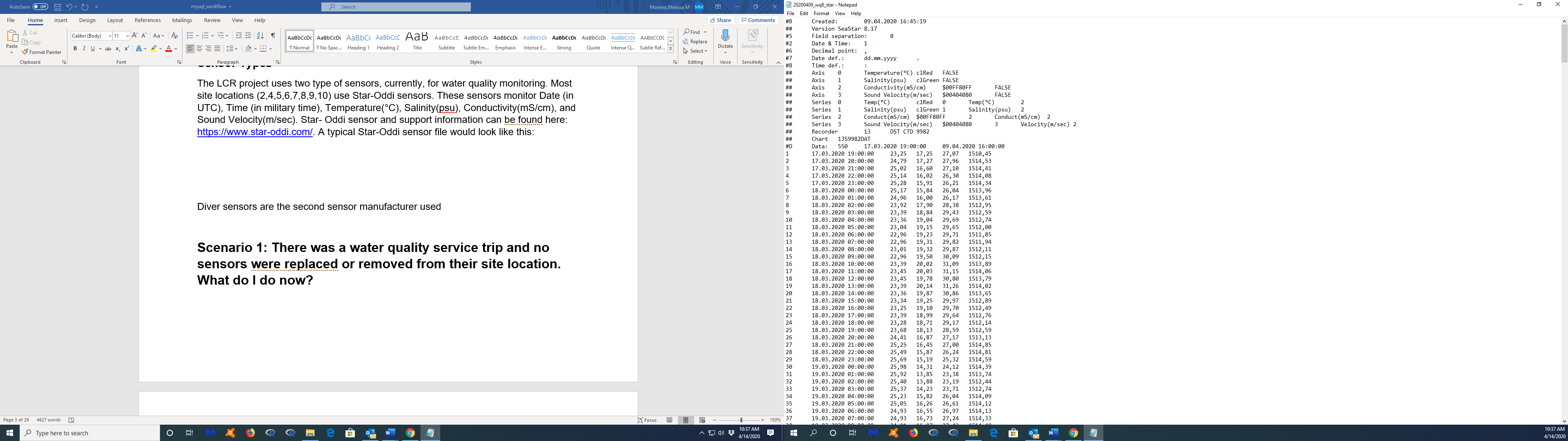


Figure 1. General overview of the MySQL workflow starting from the sensor file download in the field. Note that that the first two steps (field steps) might not be completed by the person doing the service log workflow or Python import process. This workflow will be further described in this document.

# VanEssen - Model CTD-Diver - Submersible Datalogger - Manual ...Star-Oddi | LinkedInContinuous Water Quality Sensor Types

The LCR project uses two type of sensors currently for continuous water quality monitoring. The two types are Star- Oddi sensors and Diver sensors (by vanEssen Instruments).

Star-Oddi DST CTD (3-37 mS/cm) sensors are the most used sensor in the LCR project (<https://www.star-oddi.com/products/data-loggers/salinity-logger-probe-CTD> ). Most site locations (2,4,5,6,7,8,9,10) use Star-Oddi sensors. These sensors report Date (in UTC), Time (in military time), Temperature(°C), Salinity(psu), Conductivity(mS/cm), and Sound Velocity(m/sec). The Star- Oddi sensors record these measurements once an hour, on the hour. Note salinity is not measured directly, but is calculated internally by measuring conductivity and temperature and calculating salinity from these values. Star- Oddi sensor and support information can be found here: <https://www.star-oddi.com/>. A typical Star-Oddi sensor file (.DAT) would look like this:



Screen shot- Site 8 Star- Oddi sensor file (downloaded in the field) serviced on April 9, 2020, note the name of Star- Oddi sensor file

Star- Oddi files are **required** be named in this specific format.

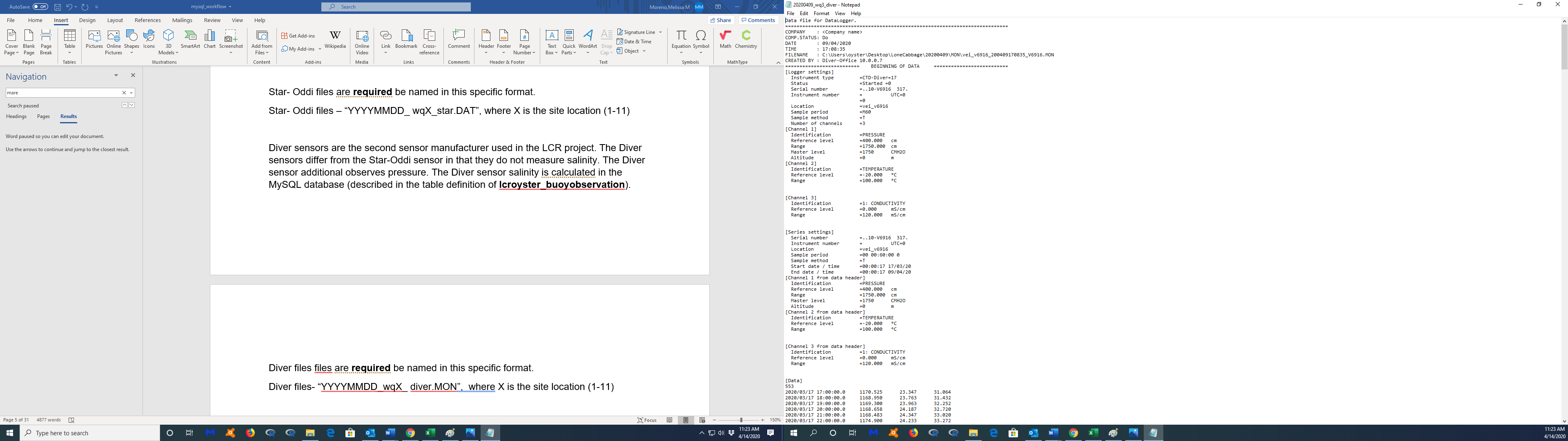
Star- Oddi files – “YYYYMMDD\_ wqX\_star.DAT”, where X is the site location (1-11)

.DAT is the file extension saved from the Star-Oddi software

Example – 20200403\_wq2\_star.DAT (Water quality site 2 Star-Oddi file downloaded on April 03,2020)

This file name is critical because the name is used as part of the MySQL import process.

Diver CTD sensors are the second sensor manufacturer used in the LCR project. Diver sensors are currently only located in water quality sites 1 and 3. The Diver sensors differ from the Star-Oddi sensor in that they do not convert conductivity measurements to salinity during the download, instead, this calculation is done within the MySQL database (described in the table definition of **lcroyster\_buoyobservation** in the column *salinity\_psu\_calculated*). The Diver sensor additionally observes pressure. Diver sensors monitor Date (in UTC), Time (in military time), Pressure (CMH2O), Conductivity (mS/cm), and Temperature (°C). The Diver sensors record these measurements once an hour, on the hour. You can refer to the Diver guide for additional sensor questions: <https://diver-water-level-logger.com/diver-water-level-loggers/ctd-diver.html>. A typical Diver sensor file would look like this:



Screen shot- Site 3 Diver sensor file (downloaded in the field) serviced on April 9, 2020, note the name of the Diver sensor file

Diver files are **required** be named in this specific format.

Diver files- “YYYYMMDD\_wqX\_ diver.MON”, where X is the site location (1-11)

.MON is the file extension saved from the Diver software

Example – 20200404\_wq1\_star.DAT (Water quality site 1 Diver file downloaded on April 04,2020)

Newly downloaded Star-Oddi and Diver sensor files need to be located in a specific folder for the Python import process to work. This file is located here:

T:\Oyster Project\oyster\_project2\project\_task\_working\t7\_data\_management\wq\data\new\_data

Based on these file names the Python import process will know how to handle and import the data from the files. Both file types have a completely different file structure, saved from their respected software, and the Python import process recognizes this based on the name of the file.

Both the Star-Oddi and Diver sensors were extensively tested before the project started using known salinity standards at a range of temperatures. Both sensors performed similarly in terms of accuracy and precision of measurements. We have used two sensor types as redundancy in case one type of sensor is not available for purchase during the entire life of the project. We have also deployed the Diver sensors at the northern and southern ends of the reef to record water pressure as a depth proxy to calculate tidal lags between observed and predicted at the Cedar Key station and at each end of the reef.



# Discrete Water Quality Sensor Types

There are two types of discrete water quality sensor types for the LCR project, Lakewatch and Yellow Springs Instrument (YSI).

Lakewatch measurements are collected in sites 1-6 on a monthly basis. The LCR project pays for the Lakewatch lab to process the water samples. Lakewatch measurements are processed and updated back to the LCR project approximately every 3 months. More information about Lakewatch can be found here: <https://lakewatch.ifas.ufl.edu/>.

Lakewatch processes the water samples collected at the LCR project sites 1-6 and returns the measurements of Total Phosphorus (µg/L), Total Nitrogen (µg/L), Chlorophyll (µg/L), Secchi (feet, recorded in the field), Color (Pt-Co Units), Specific conductance (µS/cm @ 25° C), and Specific conductance (mS/cm @ 25° C). **Lakewatch values are entered manually into the MySQL database** (described later in this document). The Specific conductance (mS/cm @ 25° C) is the value entered into the MySQL database in the **lcroyster\_waterobservation** table (this will be described later in this document).

Screen shot- Lakewatch water quality results.

The Lakewatch water quality results are e-mailed to the PIs on the LCR project. Files should be renamed:

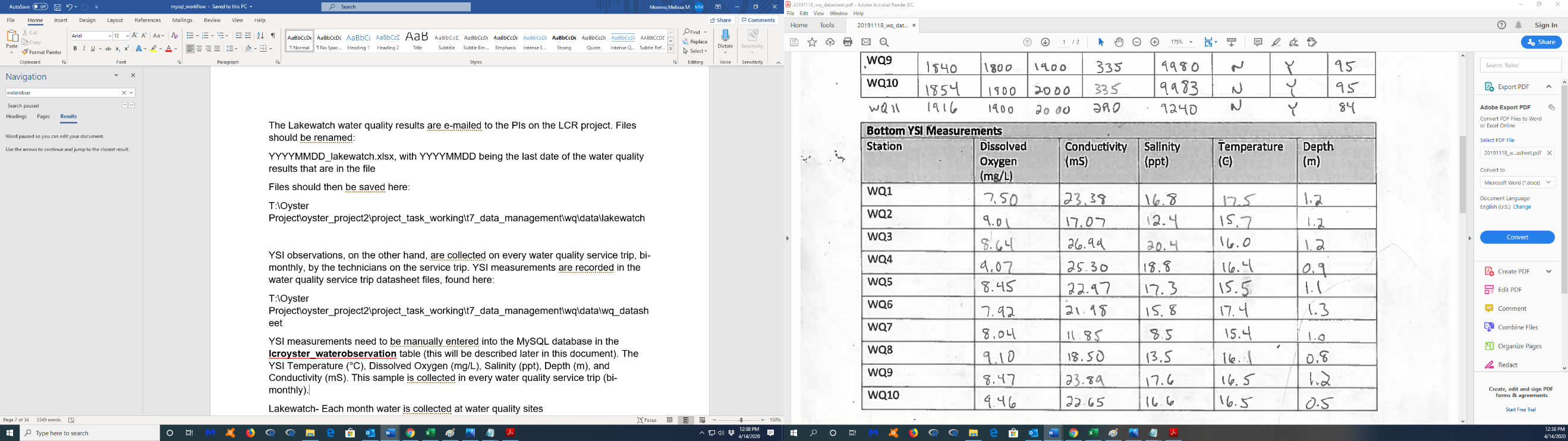
YYYYMMDD\_lakewatch.xlsx, with YYYYMMDD being the last date of the water quality results that are in the file

Files should then be saved here:

T:\Oyster Project\oyster\_project2\project\_task\_working\t7\_data\_management\wq\data\lakewatch

YSI observations are collected on every water quality service trip, bi-monthly, by the technicians on the service trip on sites 1-10. YSI measurements are recorded in the water quality service trip datasheet files, found here:

T:\Oyster Project\oyster\_project2\project\_task\_working\t7\_data\_management\wq\data\wq\_datasheet

**YSI measurements need to be manually entered into the MySQL database** in the **lcroyster\_waterobservation** table (this will be described later in this document). The YSI records Temperature (°C), Dissolved Oxygen (mg/L), Salinity (ppt), Depth (m), and Conductivity (mS). The YSI is an additional check on continuous water quality sensor measurements.

Screen shot- YSI measurements physically written in the water quality service trips

# Service Log Workflow (mandatory for every water quality service trip)

1. Enter the latest water quality service datasheet information into the service\_log.xlsx.

The service\_log.xlsx is located here:

T:\Oyster Project\oyster\_project2\project\_task\_working\t7\_data\_management\wq

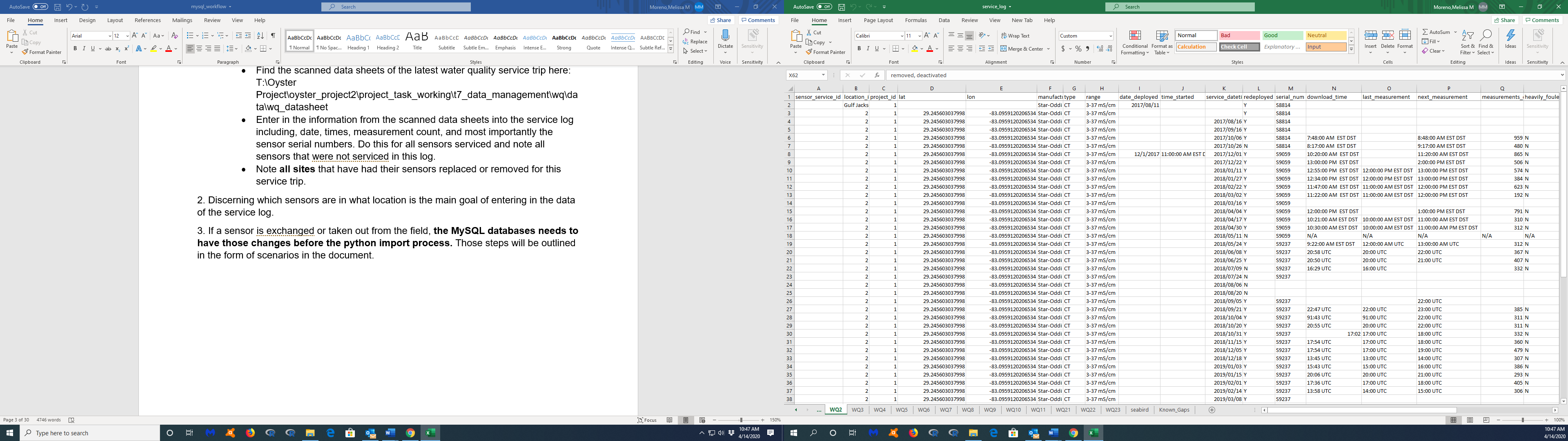
Datasheets are located here:

T:\Oyster Project\oyster\_project2\project\_task\_working\t7\_data\_management\wq\data\wq\_datasheet

2. Enter in the information from the scanned data sheets into the service log including, date, times, measurement count, and most importantly the sensor serial numbers. Do this for all sites serviced and make a note in a ‘notes` column if a sensor was replaced or removed.

* Note **all sites** that have had their sensors replaced or removed for this service trip. Note the sensor serial numbers that have been replaced or removed.

3. Discerning which sensors are in what location is the main goal of entering in the data of the service log.

* If a sensor is exchanged or taken out from the field, **the MySQL databases needs to have those changes before the python import process.** Those steps will be outlined in the form of scenarios in the document.

Screen shot- service\_log.xlsx with all Sites separated in individual worksheets (bottom)

**Scenario: There was a water quality service trip and no sensors were replaced or removed from their site location. We downloaded Star-Oddi and Diver sensor files in the field. What do I do now?**

1. First, follow the steps outlined in the Service Log Workflow (mandatory). Since no sensors were removed or replaced from their site locations, the import process is relatively easy and straight forward.

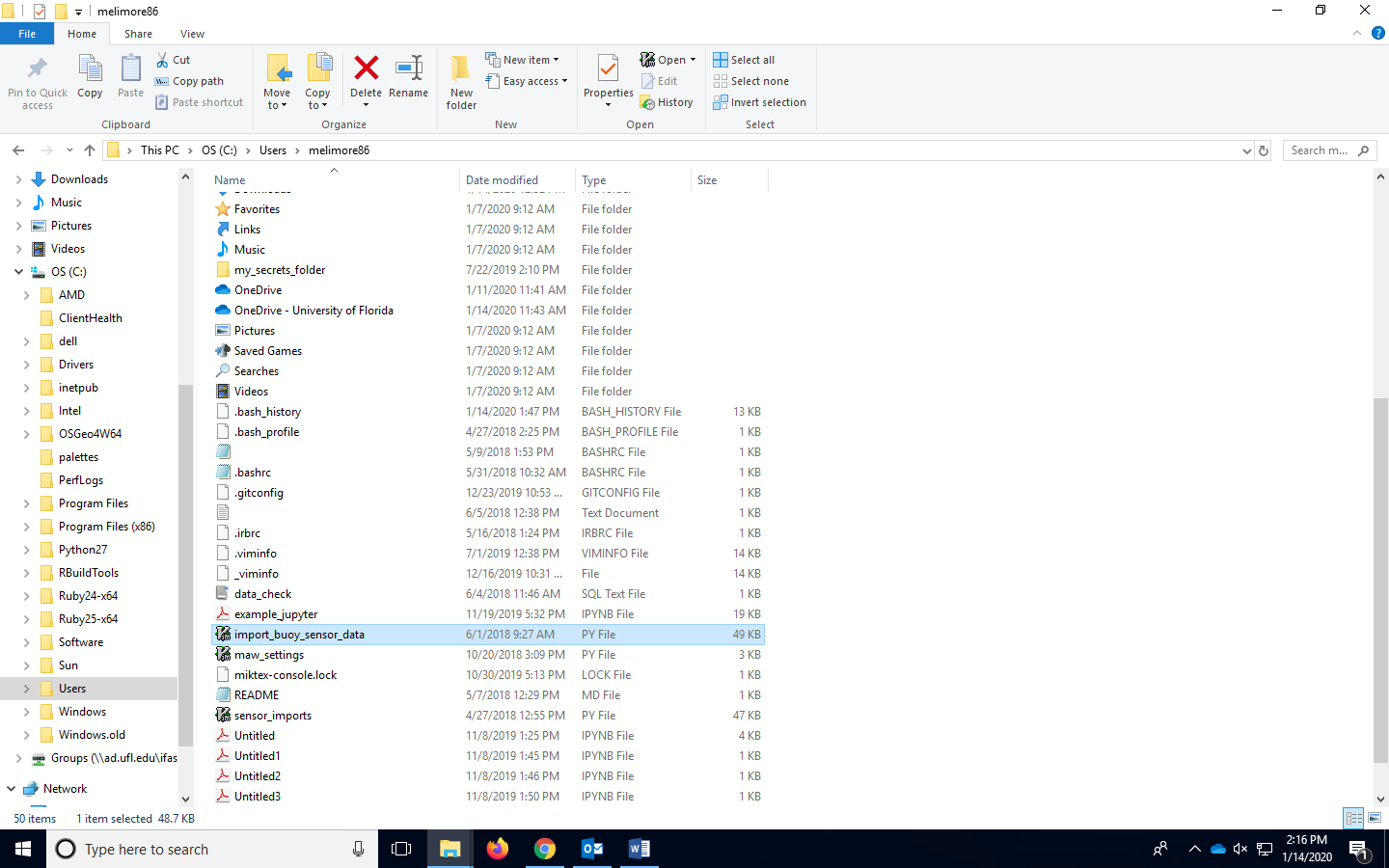
2. Check the Star- Oddi and Diver sensor files to ensure that their file names are correct in the format outlined in the **Continuous Water Quality Sensor Types** section.

- The newly field downloaded sensor files are located here:

T:\Oyster Project\oyster\_project2\project\_task\_working\t7\_data\_management\wq\data\new\_data

- Double check the filenames continuous sensor data files as per the required format in the section **Continuous Water Quality Sensor Types**. Double click on each continuous sensor data file to check the data inside. Some things to check for are proper columns names (as described in the section **Continuous Water Quality Sensor Types**) and appropriate range of dates and times.

3. Navigate to the folder where the *import\_buoy\_sensor\_data.py* and my\_secrets\_folder folder are located.



Screen shot-The location of the current *import\_buoy\_sensor\_data.py* and my\_secrets\_folder folder

* Git Bash terminal is required for the next step. This is not the GitHub online portal (github.com). This is not a Github repository. This is not the Github Desktop App. Git Bash is an emulation layer for a Git command line experience.
* You can download Git Bash terminal here: <https://git-scm.com/downloads>

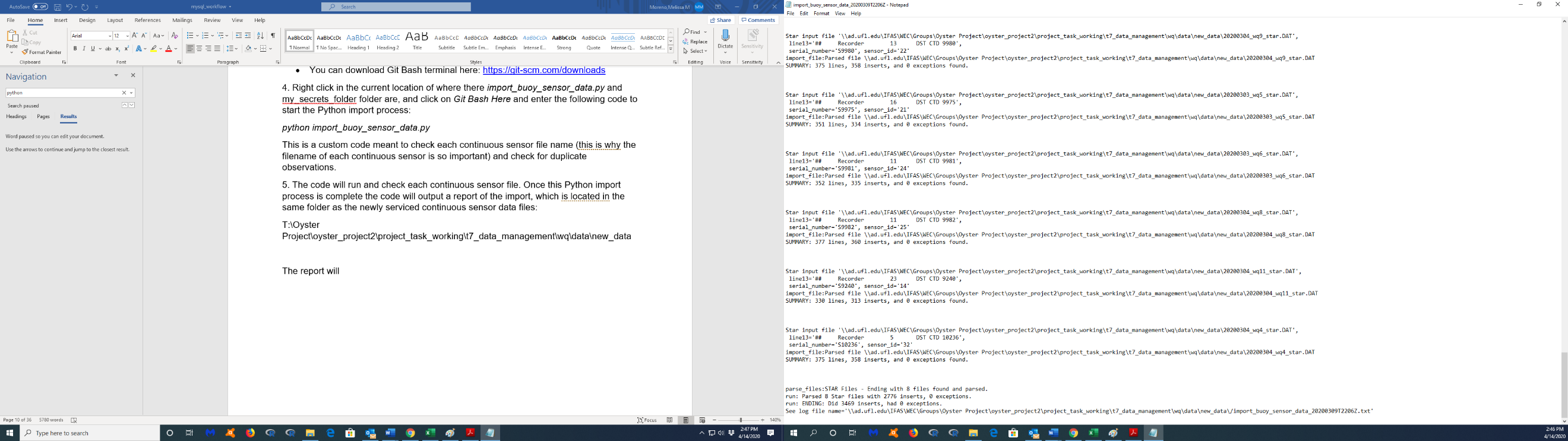
4. Right click in the current location of where there *import\_buoy\_sensor\_data.py* and my\_secrets\_folder folder are, and click on *Git Bash Here* and enter the following code to start the Python import process:

*python import\_buoy\_sensor\_data.py*

This is a custom code meant to check each continuous sensor file name (this is why the filename of each continuous sensor is so important) and check for duplicate observations. Let this code run until it’s complete, which will be when a new line to enter code is available.

5. The code will run and check each continuous sensor file. Once this Python import process is complete the code will output a report of the import, which is located in the same folder as the newly serviced continuous sensor data files:

T:\Oyster Project\oyster\_project2\project\_task\_working\t7\_data\_management\wq\data\new\_data

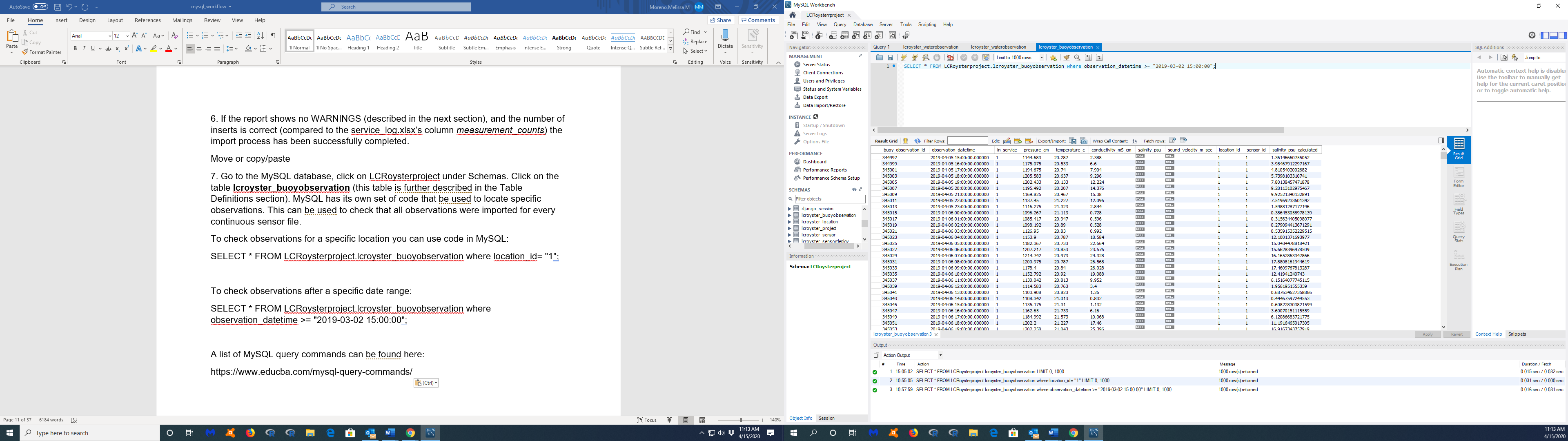
The report will give an import record of each continuous sensor file and state the number of observations imported.

Screen shot- Continuous sensor data python import process report, circled in red are the final import inserts

6. If the report shows no WARNINGS (described in the next section), and the number of inserts/observations is correct (compared to the service\_log.xlsx’s column *measurement\_counts*) the import process has been successfully completed.

Move or copy/paste the import process output report here:

T:\Oyster Project\oyster\_project2\project\_task\_working\t7\_data\_management\wq\python\_import

7. Go to the MySQL database, click on LCRoysterproject under Schemas. Click on the table **lcroyster\_buoyobservation** (this table is further described in the **Table Definitions** section). MySQL has its own set of code that be used to create a query. This can be used to check that all observations were imported for every continuous sensor file correctly.

Screen shot- Area in MySQL Workbench to query

Some examples that can be used in MySQL are:

To check observations for a specific location you can use code in MySQL:

SELECT \* FROM LCRoysterproject.lcroyster\_buoyobservation where location\_id= "1";

To check observations after a specific date range:

SELECT \* FROM LCRoysterproject.lcroyster\_buoyobservation where observation\_datetime >= "2019-03-02 15:00:00";

A list of MySQL query commands can be found here:

<https://www.educba.com/mysql-query-commands/>

8. Move or copy/paste all the continuous sensor files that have just been imported here:

T:\Oyster Project\oyster\_project2\project\_task\_working\t7\_data\_management\wq\data\imported\_data

9. Make sure the **new\_data** folder (located here T:\Oyster Project\oyster\_project2\project\_task\_working\t7\_data\_management\wq\data\new\_data) is empty. Once it is empty, this signifies that the Python import process is complete and new continuous sensor files are ready to be added to the **new\_data** folder (after the next water quality service trip).

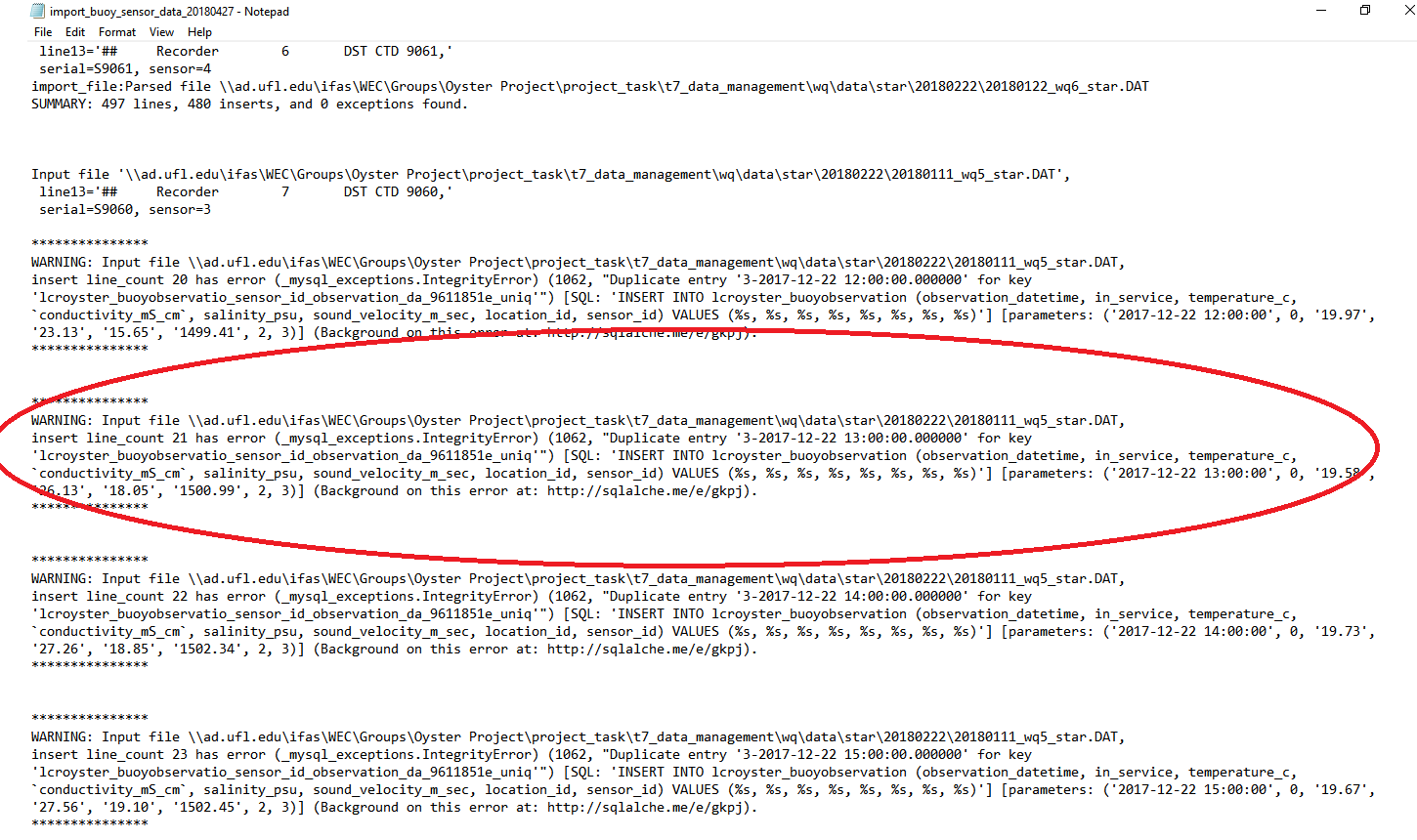
Note: All continuous sensor file observations are stored in the **lcroyster\_buoyobservation** table in MySQL Workbench based of the Python script (go to the **Table Definitions** section for descriptions of each column).

# Scenario: I did the Python import process, and I found a WARNING message in my import process output report. What do I do?

It is important to ALWAYS check the import process output report at the end of the Python import process. This report will show all the continuous sensor files, and whether the observations were either imported or not.

A WARNING message could be reported for the following reasons:

* Most common- A continuous sensor file has an incorrect filename, correct this by changing the name (check for correct file naming formats in the section **Continuous Water Quality Sensor Types**)
* Most common- A specific sensor serial number in a download file is not accounted for in the **lcroyser\_sensor** table. Review the “Scenario: The LCR project bought several new Star-Oddi/Diver sensor devices, how do I account for these new sensors in MySQL?”
* Most common- An active sensor serial number is not considered “checked-in” in the **lcroyster\_sensordeploy** table. Review the “Scenario: I replaced a sensor in the field, what do I do now?”
* Most common- A specific water quality site location is not accounted for in the **lcroyser\_location** table. Review the “Scenario: The LCR project has added new water quality sites, where do I specify this in MySQL?”
* Common- The continuous sensor files were already imported, and the WARNING is to declare a duplicate import entry, double check in MySQL Workbench and perform queries to find specific observations, if the observations were already imported the continuous sensor files might have not been moved to the correct folder location after the Python import process, check this
* Rare- Continuous sensor file observations are not in the right format, and possibly could have been corrupted during the download process, rectify this by tracking the download back to the original field computer and inspecting the download, this should not happen if the files were thoroughly checked during the field download, resolution would be to edit the continuous sensor file or not download it with the Python import process



Screen shot- An example of a WARNING message in an import process output report

# Scenario: I just imported the continuous water quality data using the python import process and there are YSI measurements from the water quality datasheet that I need to manually enter into MySQL. How do I do this?

YSI observations are what will be manually entered inside the MySQL Workbench bi-monthly, until YSI observations are no longer conducted (as per PI request). Lakewatch, as previously mentioned, are usually reported back to an LCR project PI every 3 months.

1. Go to the LCRoyster MySQL Workbench.

2. Go to the **lcroyster\_waterobservation** table (on the left-hand side of MySQL Workbench).

3. Enter in the information according to the fields specified. With every few cells that are completed, click the Apply button (bottom right button). Frequently save with the Apply button. If a field is incorrectly typed and the updates tried to be applied, this will not save. Fields will need to be reverted (“Revert”, bottom right button) back to their original state, meaning all the manual input will be lost, and the fields will need to be re-entered again. Save frequently. View the **Table Definitions** for each column description and applicable data format entry into each cell. In the *sensor\_id* column the sensor ID for YSI measurements are:

35= YSI

The time for the YSI measurements, entered into column *observation\_datetime,*are the times in the water quality service datasheet. Go back to the **wq\_datasheet** to investigate the times of the water quality observation:

T:\Oyster Project\oyster\_project2\project\_task\_working\t7\_data\_management\wq\data\wq\_datasheet

4. Apply the changes (button bottom right).

# Scenario: I was just forwarded Lakewatch results. What do I do with them?

The manual entry for Lakewatch results are virtually the same as for the YSI measurements, except that Lakewatch results have different water quality metrics tested. These results are reported back to the LCR project PIs approximately every 3 months, meaning these results will not be updated in real time.

1. Go to the LCRoyster MySQL Workbench.

2. Go to the **lcroyster\_waterobservation** table (on the left-hand side of MySQL Workbench).

3. Enter in the information according to the fields specified. With every few cells that are completed, click the Apply button (bottom right button). Frequently save with the Apply button. If a field is incorrectly typed and the updates tried to be applied, this will not save. Fields will need to be reverted (“Revert”, bottom right button) back to their original state, meaning all the manual input will be lost, and the fields will need to be re-entered again. Save frequently. View the **Table Definitions** for each column description and applicable data format entry into each cell. In the *sensor\_id* column the sensor ID for Lakewatch measurements are:

34= Lakewatch

The time for the Lakewatch results, entered into column *observation\_datetime,*are the times in the water quality service datasheet where Lakewatch samples were collected. Go back to the **wq\_datasheet** to investigate the times of the water quality observation:

T:\Oyster Project\oyster\_project2\project\_task\_working\t7\_data\_management\wq\data\wq\_datasheet

Lakewatch results are stored in the T:Drive:

T:\Oyster Project\oyster\_project2\project\_task\_working\t7\_data\_management\wq\data\lakewatch

4. Apply the changes (button on bottom right).

Review column descriptions in the section **Table Definitions**.

# Scenario: The LCR project bought several new Star-Oddi/Diver sensor devices, how do I account for these new sensors in MySQL?

It is necessary to create a sensor ID for every sensor used in the LCR project. The sensor ID number itself is an arbitrary number, however it encapsulates all important sensor information, such as the sensor’s serial number and the sensor type. Several tables are linked through specific sensor ID number (described in **Table Definitions**). Sensor IDs must be accounted for in the **lcroyster\_sensor** table prior to the Python import process. The Python import process will check this table and match the continuous water quality sensor’s data file serial number to the serial numbers in this table, and apply the appropriate *sensor\_id* to the **lcroyster\_buoyobservation** table**.**

1. Go to the LCRoyster MySQL Workbench.

2. Go to the **lcroyster\_sensor** table (on the left-hand side of MySQL Workbench).

3. The *sensor\_id* is a unique number for each individual sensor device. Currently, each new sensor has been given a chronologically ascending number, and this system is recommened.

4. Enter in the all applicable columns related to the new sensor (review **Table Definitions**).

5. The *project\_id* number is located in **lcroyster\_project** table.

6. The sensor\_type\_id is located in the **lcroyster\_sensortype** table.

7. Press Apply (bottom right) to save the changes. Applying the changes with every new sensor addition is recommended.

# Scenario: The LCR project has added new water quality sites, where do I specify this in MySQL?

All current water quality site locations are accounted for in the **lcroyster\_location** table. The **lcroyster\_location** table is not updated frequently, as the water quality sites are set (unless changed at the request of the LCR project PI’s). However, if a new water quality site location is added to the LCR project, it is mandatory to enter in the site location information into the MySQL database prior to collecting any observations from the sensor. All new locations also need to be added to the **lcroyster\_location** table prior to the Python import process. The Python import process requires specific sensors to be active in specific locations, and the site locations are defined in the **lcroyster\_location**.

1. Go to the LCRoyster MySQL Workbench.

2. Go to the **lcroyster\_location** table (on the left-hand side of MySQL Workbench).

3. Enter all applicable site location information. The *location\_id* column is not arbitrary, and the ID should be the same number as the site location (e.g., Site 1 is *location\_id* = 1). Other MySQL tables are linked together through the *location\_id* column (review **Table Definitions**).

4. Save the changes by clicking on Apply (bottom right).

Note- This table has all the GPS coordinates (in UTM’s) for the water quality site locations.

# Scenario: I replaced a sensor in the field, how do I tell MySQL about this?

Replacing a sensor in the field is a very common procedure. Sensors are replaced in site locations if they seem not to be working, they went missing, or if it is time to service them. The advantage of using the MySQL database is it has the ability to track when sensors were removed in a specific site location. The most common way to know if a sensor has been replaced in the field, is to check the water quality service trip datasheet. The field technicians are required to report on the datasheet the sensor serial numbers serviced and replaced. This is also why the first step after any water quality service trip, is to update the service log.

The procedure to replaced a sensor in a specific site in MySQL can be thought of as a “check-out” and “check-in” procedure. All active sensors are “checked-in” and all previously replaced or lost sensors are “checked-out”. Only one sensor ID can be active per site location. This is why a sensor must be first “checked-out” out of its current site location, and a new sensor added in its place. This update needs to occur prior to the Python import process. The Python import process will locate the serial number inside of the water quality sensor’s downloaded files and allocate that serial number to a *location\_id*.

1. Go to the LCRoyster MySQL Workbench.

2. Go to the **lcroyster\_sensordeploy** table (on the left-hand side of MySQL Workbench).

3. Fill in the required fields to “check-out” a sensor. Note the *sensor\_id* is a numeric value that relates to a specific sensor in the **lcroyster\_sensor** table. Make sure that the *sensor\_id* is the correct value for the “checked-out” sensor. The time “checked-out”, is the time listed in the water quality service trip datasheet. Add accurate notes in the *notes* column to why the sensor is being “checked-out”. The *location\_id* for a removed/replaced sensor is “0”.

4. Apply these changes (bottom right).

5. Fill in the required fields to “check-in” a sensor. The time that the sensor is “checked-in” to its current site location is the time listed in the water quality service datasheet. Note the *sensor\_id* is a numeric value that relates to a specific sensor in the **lcroyster\_sensor** table.The *location\_id* for a sensor replacement is the location of the water quality site. Verify in the **lcroyster\_location** table that the site is accounted for.

6. Apply these changes (bottom right).

# Scenario: I accidently deleted a whole table in MySQL, what do I do now?

The LCR project pays for support from the UF library. There is a way to revert to a previous saved backup of the MySQL database. The MySQL database backup is saved every day at midnight by the UF IT library server, so as long as the Help Desk is contacted immediately, they will be able to revert the MySQL database into a previous backup version. A table in MySQL cannot be retrieved inside of MySQL once it is deleted, there is no way to do this. There is no way to go back a step either in MySQL (aka ctrl + z). Once a command is entered in MySQL and executed, and there is no way to go back prior to the whatever the execution command did.

Note- This should not happen, as deleting a table would require a SQL code to engage this process.

To revert back to a previous’ s day MySQL, contact the UF IT Help Desk:

Help Desk support is available in person (check their hours here) and 24/7 via phone (352-392-HELP/4357) and email (helpdesk@ufl.edu), <http://helpdesk.ufl.edu/>.

# Recommendations: MySQL tables to become familiar with

* **lcroyster\_buoyobservation** – continuous sensor observations (import using Python)
* **lcroyster\_location**- the information for the physical locations of where sensors could be found
* **lcroyster\_sensor** – information pertaining to each water quality sensor
* **lcroyster\_sensordeploy** – an account of when and where the sensors have been removed or replaced (needs to be review prior to python import)
* **lcroyster\_sensorservice** – similar information as the service\_log.xlsx
* **lcroyster\_sensortype**- unique identifier for each sensor type
* **lcroyster\_waterobservation** – discrete observations (manually entered)

Each table is specifically linked to other tables through foreign keys (columns). This document will discuss which foreign keys are link different tables.

# Table Definitions (listed in the order in the MySQL Workbench)

## Table: lcroyster\_buoyobservation

Function: This table include all continuous observations from Star-Oddi and Diver sensors. The function of this table is to store these observations. The Python import process will fill in the required fields. Little to no manual editing should be done in this table. This table will include all observations, even flat lined or questionable observations.

Column definitions:

*buoy\_observation\_id*- This field is automatically filled from the Python import process. Each observation will have its own *buoy\_observation\_id*. This number is generated chronologically.

*observation\_datetime*- All Star-Oddi and Diver sensors are in UTC. These fields will be automatically be imported in the format YYYY-MM-DD HH:MM:SS.000000.

*in\_service*- All sensor observations imported will be *in\_service,* unless otherwise specified in the **lcroyster\_sensordeploy** table, such as if the sensor is “checked-out” during the date/time specified in that table.

*pressure\_cm*- Pressure is a variable measured only by Diver sensors, currently in site locations 1 and 3. Star-Oddi sensors will have his field missing from their hourly observations.

*temperature\_cm*- All sensors record temperature measurements in Celsius.

*conductvitiy\_mS\_cm* - All sensors record temperature measurements in mS/cm.

*salinity\_psu­*- Only Star-Oddi sensors observe salinity. Diver sensors will have this field missing in their hourly observations.

*sound\_velocity\_m\_sec* - Star-Oddi sensors measure these observations. Diver sensors will have this field missing in their hourly observations because Diver sensors do not record these data.

*location\_id –* Based on the **lcroyster\_sensordeploy** table, this field will be generated based on which *sensor\_id*’s are “checked-in” to a *location\_id*. The column *location\_id* references other tables, and because of the continuity of this column, a sensor is able to be tracked in all of the *location\_id*’s it has inhabited.

This column is linked to several tables, including:

* **lcryoster\_location**
* **lcroyster\_sensordeploy**
* **lcroyster\_sensortype**
* **lcroyster\_waterobservation**
* **lcroyster\_sensorservice**

*sensor\_id*- Numerical value created in the **lcroyster\_sensor** table. This is a numeric value for a specific sensor. This column is linked to other tables:

* **lcroyster\_sensor**
* **lcroyster\_sensordeploy**
* **lcroyster\_sensorservice**
* **lcroyster\_waterobservation**

*salinity\_psu\_calculated-* This field is calculated based on the R package “marelac” equation below. This conversion is needed because the sensors actually measure conductivity and we convert these measurements of conductivity to salinity using the equation below. The R package cran documentation is located here <https://cran.r-project.org/web/packages/marelac/marelac.pdf>.

The R code reads as followed:

standard=42.914

salinity <- convert\_RtoS(conductivity /standard,

t= temperature, p= 0)

p=0 is used because we are not deploying the sensors in deep water

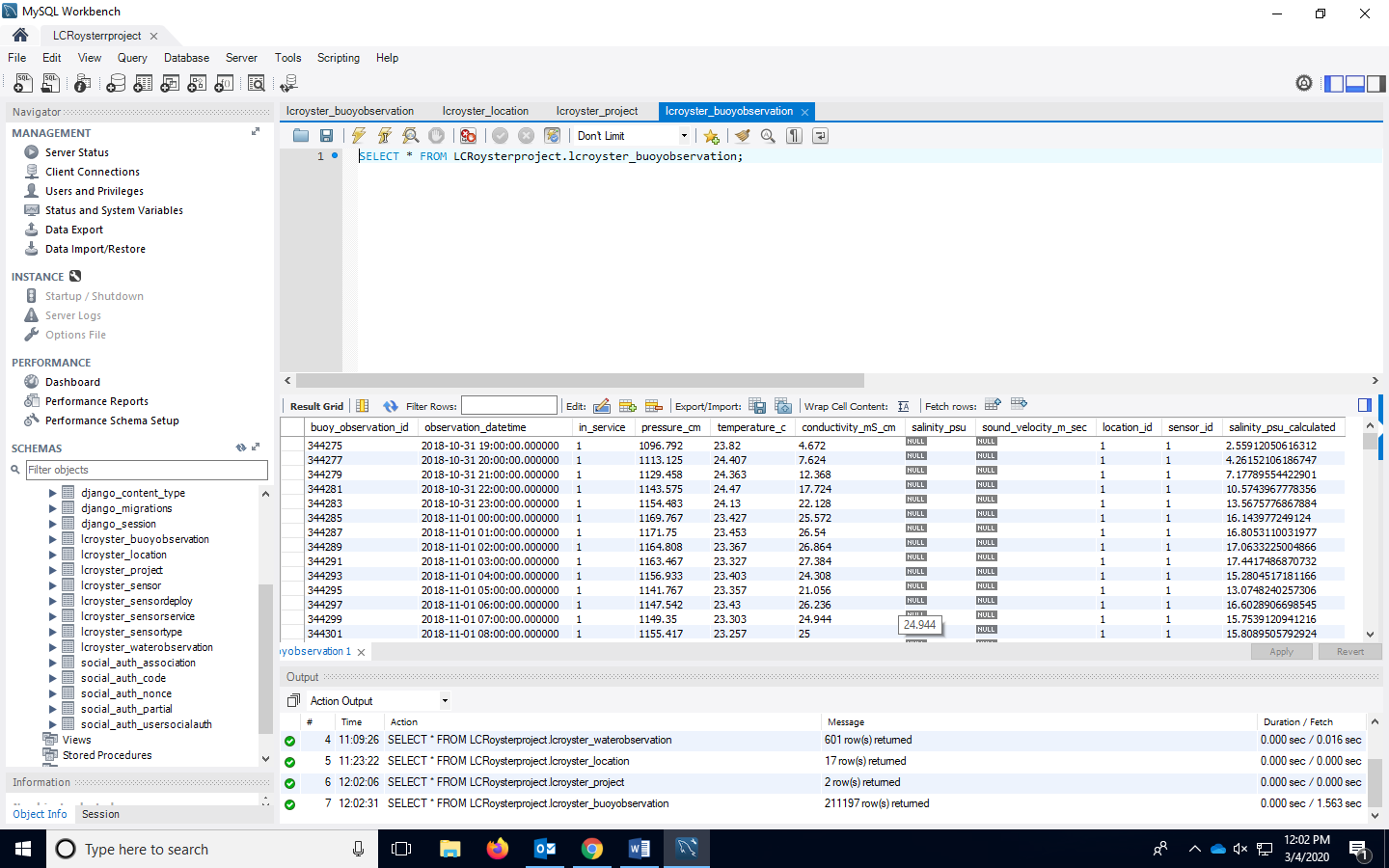
This is the salinity that is used for analysis, not the column *salinity\_psu* column because Diver sensors will not have this field filled, since those sensors do not measure salinity. The Star sensors report salinity as a field, but this is converted from the conductivity measurements.

Figure- Screen shot of **lcroyster\_buoyobservation** table.

## Table: lcroyster\_location

Function: To record all information on the physical location of a site. **Prior to the Python import process, the location of a site will need to be recorded in this table.** All the fields in this table will need to be entered, this table is not auto-generated by any import process.

Column definitions:

*location\_id-* This field will be manually entered. For readability, the *location\_id* is the same number as the physical site location name. This column influences other tables, including:

* **lcryoster\_location**
* **lcroyster\_buoyobservation**
* **lcroyster\_sensortype**
* **lcroyster\_waterobservation**
* **lcroyster\_sensorservice**

*tile\_id­­*- Originally, it was theorized to have all site locations attached to a tiling system in a map. However, this effort is still in progress, and there are no *tile\_id*’s associated with site locations at this time.

*latitude-* Latitude of the site location in decimal degrees UTMs.

*longitude*- Longitude of the site location in decimal degrees UTMs.

*name-* Common name of site location.

*alias1-* Enter an alias if one is needed for the site location.

*alias2-* Enter an alias if one is needed for the site location.

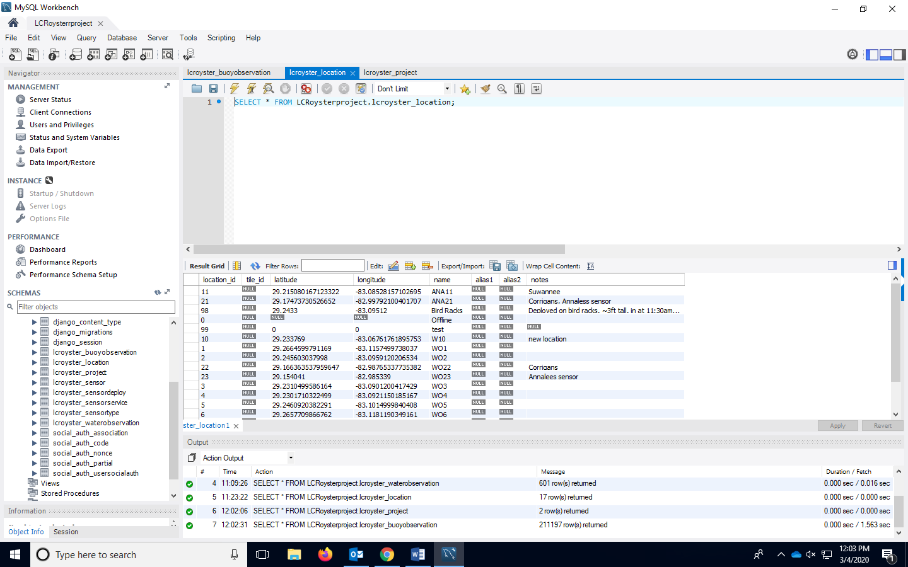
*notes­*- Generally, any notes that pertain to the creation or modification of a site should be added here.

Figure- Screen shot of **lcroyster\_location** table.

## Table: lcroyster\_project

Function: This table is to describe different projects that are contributing to the database. Almost all observations pertain to the Lone Cabbage Reef Restoration project. This table is manually entered and not automatically generated.

Column definitions:

*project\_id –* Numerical value. This field is created in this table. This field is needed for other tables:

* **lcroyster\_sensor**
* **lcroyster\_sensorservice**

*project\_website\_code*- Code created by the project manager. Normally just a short acronym.

*name­*- Name of the project.

*uf\_award\_id*- If applicable, enter an award ID.

*sponsor\_names –* If applicable, enter the names of the sponsors funding the grant.

*sponsors\_award\_id –* If applicable, enter the sponsor award ID.

*contact\_investigator*- Enter in the name of the main contact investigator of the project.

*principal\_investigators­*- The names of all PIs should be entered in this field.

*co\_principal\_investigators*- If applicable enter the names of the co PIs.

*collaborators*- If applicable, enter the names of collaborators. This can be agencies, universities, or individuals.

*award\_start\_date* – Field available to enter the start date of the grant using the format YYYY-MM-DD HH:MM:SS.000000.

*award\_end\_date* - Field available to enter the end date of the grant using the format YYYY-MM-DD HH:MM:SS.000000.

*responsible\_unit*-

*department\_id* – If applicable, enter the department ID responsible for the grant.

*proposal\_id ­*­- If applicable, enter the proposal ID responsible for the grant.

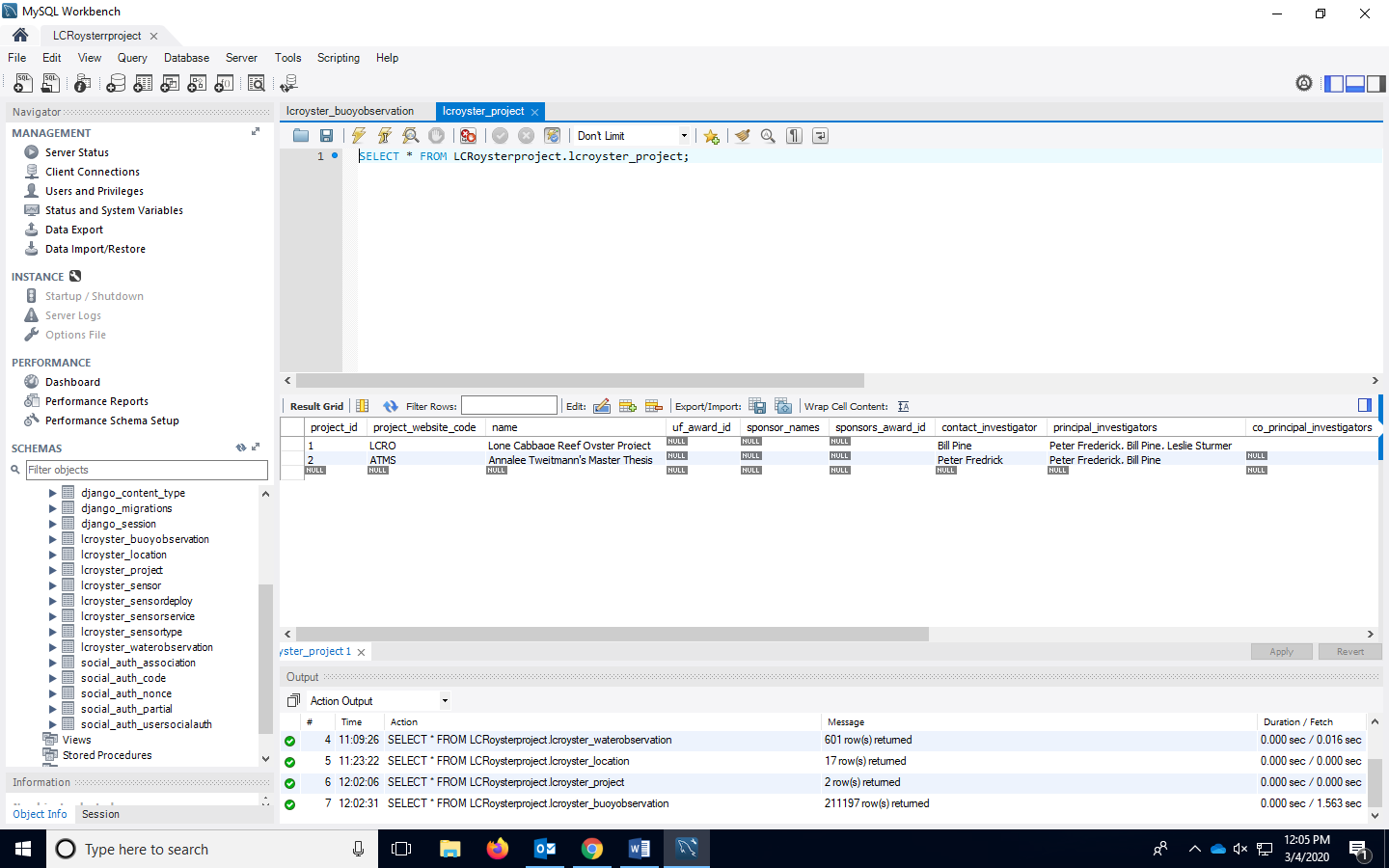
*notes ­– Enter notes applicable to the project if there is not a field that covers the information needed for the project.*

Figure- Screen shot of **lcroyster\_project** table.

## Table: lcroyster\_sensor

Function: A table to record all sensors (active, deactivated, broken, etc) that have ever been used in data collection, where its data is stored in the MySQL databased. This table is manually entered and is not auto generated. **All sensors must be listed in this table prior to the Python import process.** When a new sensor is added, a new entry is needed in this table. Make sure to add all applicable new sensors to this table before the python import process.

Column definitions:

*sensor\_id –* This column is a numerical value, and is normally chronological, but can be any numerical value. The *sensor\_id* column related to other tables:

* **lcroyster\_sensordeploy**
* **lcroyster\_sensorservice**
* **lcroyster\_buoyobservation**
* **lcroyster\_waterobservation**

Because the *sensor\_id* is consistent among all tables. Sensors are identified through this ID through the tables in the database.

*serial\_number ­–* This column is to enter the serial number of the sensor being added. All Star-Oddi sensors will have an “S” in front of their serial number. This number related to the Star-Oddi data files (for import) and needs to be exact. The Diver sensors will have a “V” in the beginning of their serial numbers, this number also has to be exact.

*range\_low\_mS\_cm*- Field to enter the low range of Star-Oddi sensors. This field might not be necessary for Diver sensors.

*range\_high\_mS\_cm* -Field to enter the high range of Star-Oddi sensors. This field might not be necessary for Diver sensors.

*manufacture\_date ­–* If applicable, this field is to enter the manufacturer creation date. This information might not always be available.

*battery\_expiration\_date-* If applicable, this field is to enter the manufacturer battery experiation date. This information might not always be available.

*notes-* Any notes that could be applicable to adding the sensor, which is not covered in the columns.

*project\_id*- This column will indicate the project in which the sensor being added, belongs to. This field is created in **lcroyster\_project** table and used to identify the project tied to each sensor.

*sensor\_type\_id* – This ID is created in the **lcroyster\_sensortype** table. Enter the corresponding sensor type numerical value based on the **lcroyster\_sensortype** table. This column is linked to other tables:

* **lcroyster\_sensor**
* **lcroyster\_sensordeploy**
* **lcroyster\_sensorservice**

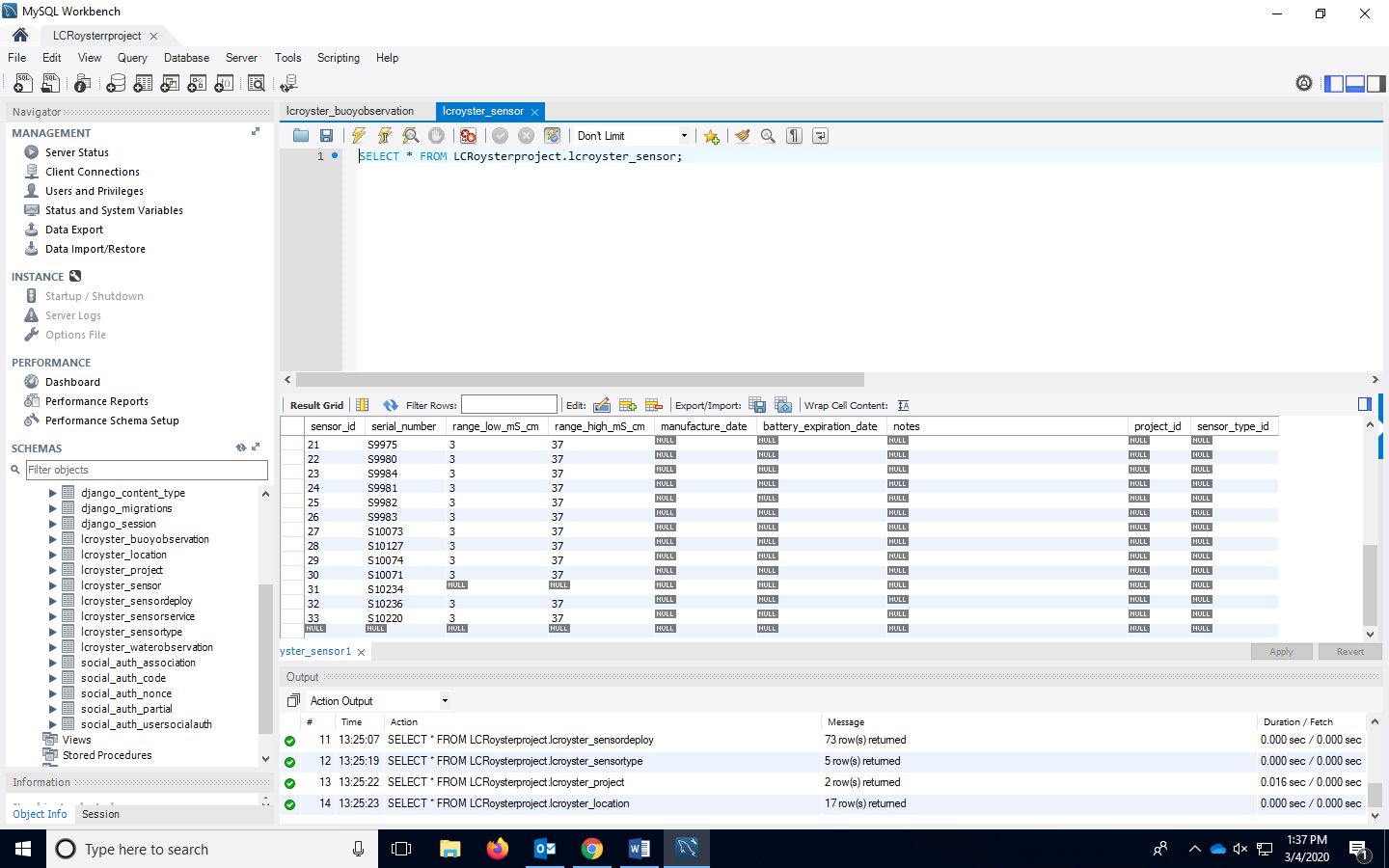


Figure- Screen shot of **lcroyster\_sensor** table.

## Table: lcroyster\_sensordeploy

Function: As per the “check-out” and “check-in” procedure, this table records all instances where a sensor has been replaced or removed from a site location.

Column definitions:

*sensor\_deploy\_id­-* Chronological numerical value.

*deploy\_datetime*- Date and time of sensor removed and/or replaced in the format YYYY-MM-DD HH:MM:SS.000000. Review the “check-out” and “check-in” scenario.

*notes*- Add any additional notes on why the sensor has been removed or replaced.

*location\_id -* Based on the **lcroyster\_sensordeploy** table, this field will be generated based on which *sensor\_id*’s are “checked-in” to a *location\_id*. The column *location\_id* references other tables, and because of the continuity of this column, a sensor is able to be tracked in all of the *location\_id*’s it has inhabited. If a sensor is being removed place “0” (zero) in the *location\_id* field.

This column is linked to several tables, including:

* **lcryoster\_location**
* **lcroyster\_sensortype**
* **lcroyster\_waterobservation**
* **lcroyster\_sensorservice**

*sensor\_id -* Numerical value originating from the **lcroyster\_sensor** table. This is a specific value for a specific sensor. Enter the numerical value that represents the specific sensor.

This column is linked to other tables:

* **lcroyster\_buoyobservation**
* **lcroyster\_sensor**
* **lcroyster\_sensorservice**

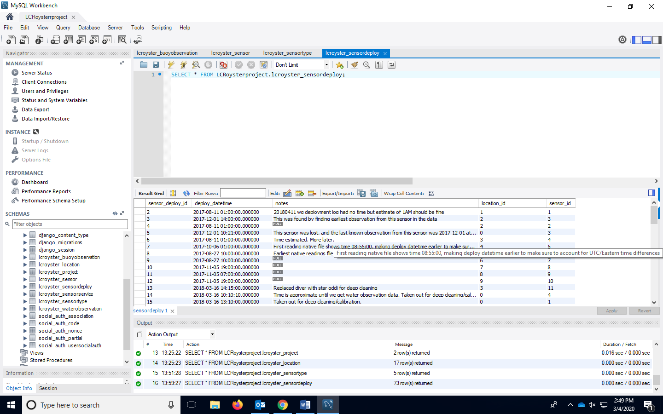


Figure- Screen shot of **lcroyster\_sensordeploy** table.

## Table: lcroyster\_sensorservice

Function: The function of this table is to be a back-up of the service\_log.xlsx. This table could be analyzed, but it is currently used to record water quality service trips. This table is manually entered and not automatically generated. **All fields in this table will need to be filled before clicking the Apply button to save changes.**

Column definitions:

*sensor\_service\_id*- Chronological numerical value.

*location\_id –* Site location that was serviced. Based on the **lcroyster\_sensordeploy** table, this field will be generated based on which *sensor\_id*’s are “checked-in” to a *location\_id*. This column is linked to several tables, including:

* **lcryoster\_location**
* **lcroyster\_sensortype**
* **lcroyster\_waterobservation**
* **lcroyster\_sensor**

*service\_datetime*- Date and time, in UTC, when the sensor was physically serviced. This needs to be in the format YYYY-MM-DD HH:MM:SS.000000.

*download\_datetime* Date and time, in UTC, when the sensor data were downloaded. This needs to be in the format YYYY-MM-DD HH:MM:SS.000000.

*active*- Refers to is a sensor was redeployed. The number “1” refers to redeployed and active, and “0” (zero) refers to a sensor not being redeployed and in active.

*measurements\_downloaded*- The number of measurements downloaded in the service trip from the particular sensor. This is a numerical value.

*heavily\_fouled*- Refers to if the sensor being serviced seemed to be heavily fouled by barnacles or other sea creatures. The number “1” refers to the sensor being heavily fouled and the number “0” (zero) refers to a sensor not being shown to be heavily fouled.

*battery\_life\_remaining\_percent­-­ ­­*Each sensor’s remaining battery life is recorded on each service trip. Enter the

*notes*- Notes on the service trip, which are not covered by any of the columns in the table.

*project\_id-* Numerical value. This field is created in this table. This field is needed for another linked table:

* **lcroyster\_sensor**
* **lcroyster\_sensorservice**

*sensor\_id-* Numerical value created in the **lcroyster\_sensor** table. This is a specific value for a specific sensor. This column is linked to other tables:

* **lcroyster\_sensordeploy**
* **lcroyster\_sensor**
* **lcroyster\_buoyobservation**
* **lcroyster\_waterobservation**

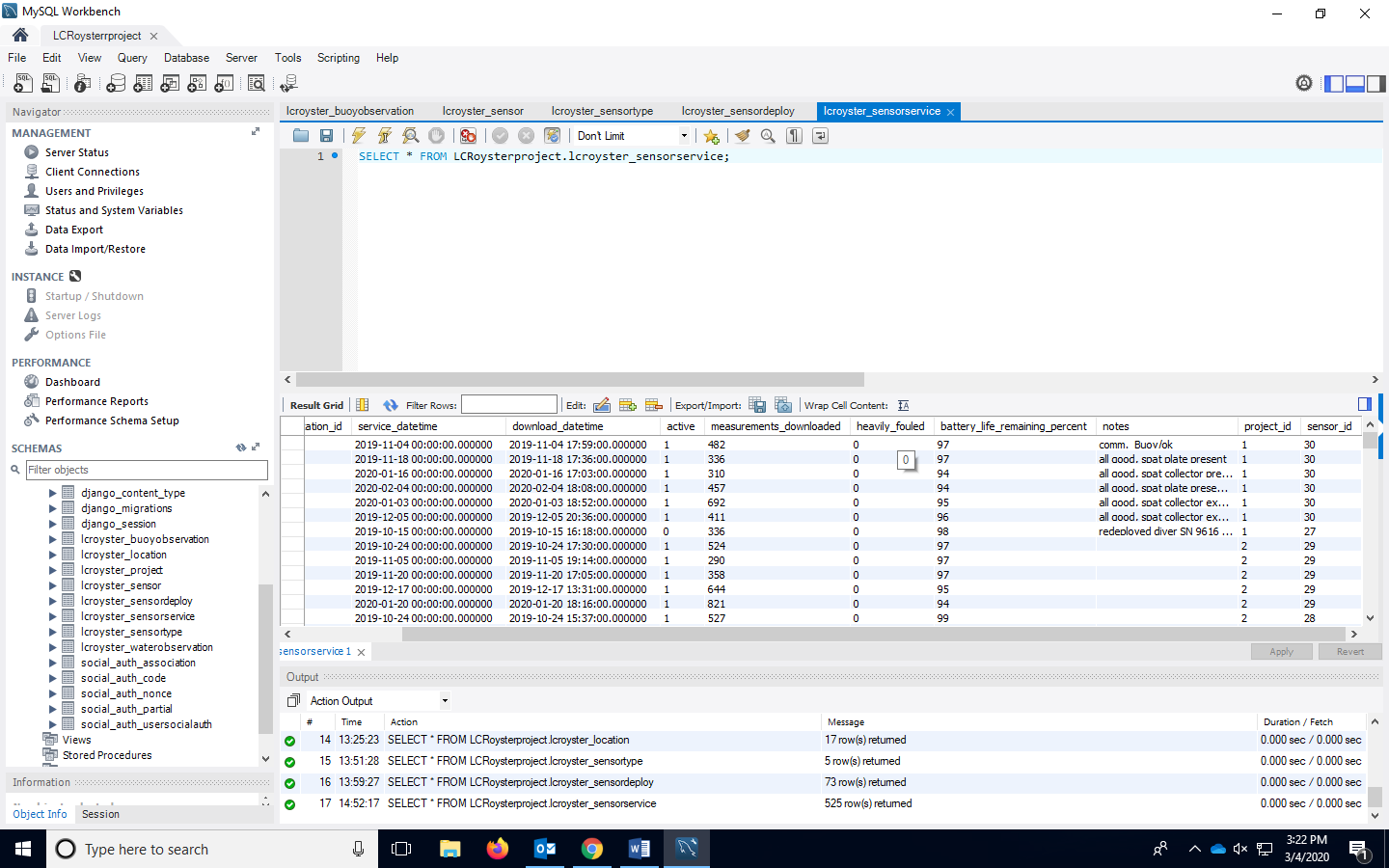


Figure- Screen shot of **lcroyster\_sensorservice** table.

## Table: lcroyster\_sensortype

Function: This table describes the sensor type for all sensors. This table specifies if one sensor type is a Star-Oddi or a Diver. Other sensor types are listed in this table and need to be account for if their data will be stored inside the MySQL database. This table is manually entered and not automatically generated.

Column definitions:

*sensor\_type\_id* – This value is numeric and is to reference a sensor type. For example, discrete YSI values are a *sensor\_type\_id* of “5”. Every observation with this sensor ID will be attributed to having been collected through a YSI instrument.

This column is linked to other tables including:

* **lcroyster\_sensor**
* **lcroyster\_waterobservation**

*manufacturer-* This column is to enter the manufacturer of the sensor.

*model\_type –* This column is to reference the model of the sensor.

*notes-* Write any notes that pertain to the sensor that are not covered in any of the columns.

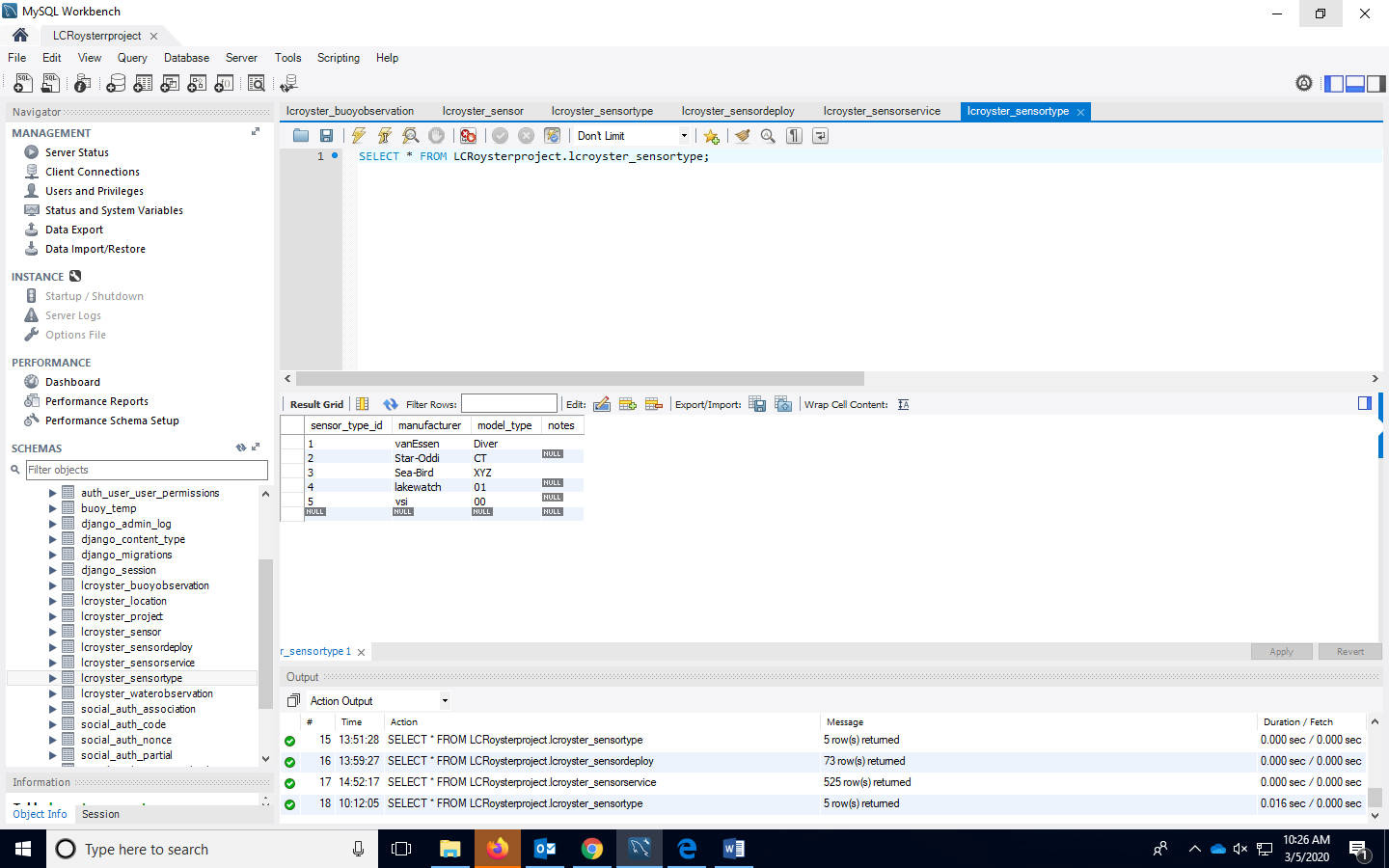


Figure- Screen shot of **lcroyster\_sensortype** table.

## Table: lcroyster\_waterobservation

Function: The function of this table is to store discrete observations/measurements from an instrument, such as a YSI. This table is manually entered and not automatically generated.

Column definitions:

*water\_observation\_id –­* Chronological numerical value for each observation. Ascending numerical value per each new observation.

*observation\_datetime -* Enter it in the format of “YYYY-MM-DD HH:MM:SS.000000”. Use the time entered in the service\_log.xlsx for the YSI (*sensor\_id* = “35”) discrete measurement. Lakewatch measurement (*sensor\_id*= “34”)

*phosphorous\_ug –* Measurement from Lakewatch analysis in units µg/mL. This parameter is not measured by the YSI.

*nitrogen\_ug-* Measurement from Lakewatch analysis in units µg/mL. This parameter is not measured by the YSI.

*chlorophyll\_ug -* Measurement from Lakewatch analysis in units µg/L. This parameter is not measured by the YSI.

*secchi\_ft* – Secchi (depth) measured by Lakewatch in feet. This parameter is not measured by the YSI.

*color\_pt\_co -*Measurement from Lakewatch analysis in units Platinum-Cobalt Scale. This parameter is not measured by the YSI.

*temperature\_c –* Temperature is measured in the field by the YSI, in Celsius.

*conductivity\_mS\_cm* – Both the YSI and Lakewatch analysis will have this measurement. Conductivity is in the unit mS/cm.

*location\_id –* The water quality site location ID of where the YSI or Lakewatch sample was collected. The column *location\_id* references other tables, and because of the continuity of this column, a sensor is able to be tracked in all of the *location\_id*’s it has inhabited.

This column is linked to several tables, including:

* **lcryoster\_location**
* **lcroyster\_sensordeploy**
* **lcroyster\_sensortype**
* **lcroyster\_sensorservice**

*sensor\_id -* Numerical value created in the **lcroyster\_sensor** table. This is a specific value for a specific sensor. This column is linked to other tables:

* **lcroyster\_sensor**
* **lcroyster\_sensordeploy**
* **lcroyster\_sensorservice**
* **lcroyster\_buoyobservation**

35= YSI

34= Lakewatch

*salinity\_psu –* Salinity measured in the field by the YSI in units ppt/ psu. This is not measured by Lakewatch analysis.

*do\_mgl –* Dissolved oxygen measured in the field by the YSI in units mg/L. This is not measured by Lakewatch analysis.

*depth\_m* – Measured in the field with the YSI in units meters.



Figure- Screen shot of **lcroyster\_waterobservation** table.

# Resources at UF:

Joe Authmuth

* Associate University Librarian
* Office: L-301A Marston Science Library
* Phone: (352) 273-0371
* Email: mapper@uflib.ufl.edu

Dan Maxwell

* Associate University Librarian
* Office: L-301B, Marston Science Library
* Phone: (352) 294-1034
* Email: danielmaxwell@ufl.edu

Plato Smith

* Data Management Librarian
* Phone: (352)273-8399
* Email: [plato.smith@ufl.edu](mailto:plato.smith@ufl.edu)

Addition MySQL Online Resources:

MySQL Crash Course | Learn SQL -<https://www.youtube.com/watch?v=9ylj9NR0Lcg>

MySQL Tutorial for Beginners Learn in 7 Days- <https://www.guru99.com/mysql-tutorial.html>

MySQL Tutorial - <https://www.tutorialspoint.com/mysql/index.htm>