# Practical Assignment 4.1 – Detailed Instructions

The purpose of this assignment is to demonstrate the process of designing and creating a database from raw data. In a perfect world, data would always be organized in a way conducive to use in a variety of database management systems—however, this is rarely the case. In this exercise, we begin unformatted raw data, and will convert this data for use in a relational database (MariaDB… although this exercise would be relevant for nearly any DBMS).

The basic process to convert the data consists of analyzing the raw data and (if we are lucky) a data dictionary, then building a normalized relational schema, and finally importing the data from its raw format into the structured database.

There are many tools to complete the various tasks, however, for this assignment I will demonstrate using commonly available tools (Microsoft Excel, PHP, MySQL Workbench, etc.) For the scripting tasks, I have chosen to use PHP. This is simply a personal preference, and that most students at Drexel enter my course with proficiency in either PHP or Python. However, these tasks could be completed in any language of your preference; you are welcome to complete the assignment using a different language with which you are more familiar.

Note: There are four appendices in this document, and it may be helpful to review them before beginning the lab:

* [Appendix A: Using Notepad++](#_Appendix_A:_Using)
  + Notepad++ has a built in SSH/SFTP client, that will let you edit script files (such as SQL and PHP) directly on your server. It also has format highlighting that some students find helpful.
* [Appendix B: Using the SCP command](#_Appendix_B:_Using)
  + SCP allows you to copy files to/from your remote Linux server using a simple command, which is available in PowerShell in Windows and in the console on Mac. This will be useful for uploading larger files, like CSVs
* [Appendix C: Install / Test PHP](#_Appendix_C:_Install)
  + We will be using PHP; if you are not sure how to install PHP on CentOS, follow these instructions.
* [Appendix D: Learn PHP](#_Appendix_D:_Some)
  + In this lab, I will provide all the code you need… but most students find it helpful to understand PHP in case they want to make changes or debug the code.
  + If you already know Python, Perl, C, C++, JavaScript, etc… learning PHP will be rather easy. Skim the tutorial to familiarize yourself with the syntax of PHP
  + If you are not familiar with programming… The PHP tutorial on W3Schools will help you learn enough proficiency to easily complete this lab. This appendix is the sections of the W3Schools tutorial relevant to this lab, with some adjustments.

There are several parts to this assignment:

1. [Analyze Raw Data](#DataAnalysis)
   1. We look at some raw data from data.gov, and data dictionaries provided by them
   2. We devise a strategy for developing a normalized database to support this data and the business need
   3. Enabling “information” from “data”
2. [Create User Tables](#CreateUserTables)
   1. We will use two traditional methods to import some simple data tables:
      1. MysqlImport utility
      2. Using MySQL Workbench
   2. [Deliverable One](#Deliverable1)
3. [Prepare Schema Definition Files](#PrepareSchemaDefinition)
   1. File One: Supporting tables (derived from analyzing the data and dictionary)
   2. File Two: Data Files
4. [Upload Dictionary and Data Files](#UploadCSV)
5. [Create Supporting Tables](#CreateSupportingTables)
   1. Write a PHP script that uses the Data Dictionary file for supporting tables to create a SQL script
   2. Write a PHP script that uses the Data Dictionary file for supporting tables to generate SQL insert statements
   3. Execute the scripts using the mysql command line utility
   4. [Deliverable Two](#Deliverable2)
6. [Create Data Tables](#CreateTables)
   1. Write a PHP script that uses the Data Dictionary file to create the data tables
   2. Execute the script using the mysql command line utility
7. [Import Data](#ImportData)
   1. Write a PHP script that will import the data from the data CSV into the appropriate tables.
8. [Use MySQL Workbench to generate an E-R diagram of your database.](#Diagram)
   1. [Deliverable Three](#Deliverable3)
9. [Test your database using queries that include JOIN statements](#TestQueries)
   1. [Deliverable Four](#Deliverable4)

### Scenario:

In this fictious scenario, you have been asked to create a database for information about US colleges which will be used by financial institutions with third party applications. A web service API will be developed, which will allow other applications to query the college information, for tasks such as information validation, searching/populating college information, underwriting decisions, etc.

Instructions:

1. **Analyze the Raw Data**
   1. For this assignment, we will be using data from the US Government’s [Open Data](http://data.gov) website. This is a public repository of data made available by the many government entities in the United States (not just the Federal Government). You can find some rather interesting datasets here, and you could repeat this assignment with many of them. However, for this course, we will be using the [Department of Education’s College Scorecard](https://catalog.data.gov/dataset/college-scorecard). This is a dataset that contains information about every college in the United States for which the Department of Education approves Federal student loans and grants.
   2. You will need the following two files:
      1. 2016-17 College Scorecard Data (CSV)
         1. 
      2. Data dictionary that describes the data columns/values
         1. 
   3. Open the **MERGED2016-17\_PP.csv** file and familiarize yourself with its contents.
   4. Open **CollegeScorecardDataDictionary.xslx** and note that it describes the data.
      1. We will use this later, to help design what tables we will need, their relationships, primary/foreign keys, etc.
2. **Create User Tables**
   1. In our example, since this data will be used through web service APIs, we would need a set of tables to store account information (*such as API Keys, and information about the applications accessing them*).
   2. The developer has provided some sample data in a spreadsheet, attached here:



* + 1. Note, that while each tab has values for the obvious data elements, there are no primary keys; we will need to create one. On each tab, add a new column (Column A) called ID, and populate each row with a unique integer:



* + - * 1. Save this table as **APIUsers.txt**



* + - * 1. Save this table as **APIUserDetail.csv**
  1. Create the CollegeScorecard database in your MariaDB Server:
     1. Login to your Linux VM, using SSH. On Windows, you can use PuTTY, or the [**SSH command in PowerShell**](https://youtu.be/Je7BILSaBSY) (*which will not require the installation of any application*)
     2. Make sure you have a “root” prompt (“**#**”, not “**$**” proceeding the cursor position). To get to a root prompt, use the “**su**” command, and provide your root password. (*Note: you will need to be using the root session for most of the assignments in this course*).
     3. Log in to MariaDB as root… Use the command “**mysql -uroot**”
        1. *If you have configured a root password in MariaDB previously, you will need to use the command* “**mysql -uroot -p**” *and provide your MariaDB root password.*
           1. NOTE: the root account in MariaDB is not the same as the root account in Linux. Although they may have the same password, this is not always the case. We will learn more about database user accounts later.
        2. You should now be connected to MariaDB… use the **SHOW DATABASES;** command and note the databases in your MariaDB instance.
        3. Use the **DROP DATABASE test;** command to remove the test database from your MariaDB instance.
        4. Create your new database for the CollegeScorecard using the command: **CREATE DATABASE CollegeScorecard;**
           1. Note: you should now see the new database when executing the **show databases** (or **show schemas**) command.
  2. Create the API User Tables:
     1. In the mysql utility database prompt, type “**use CollegeScorecard**” to enter the CollegeScorecard database. Note, that the name of the database (and database objects in general) are case sensitive.
     2. Use the “**CREATE TABLE**” command to create a table for APIUSers:

**CREATE TABLE APIUsers (**

**ID INT NOT NULL AUTO\_INCREMENT,**

**UserID VARCHAR(64) NULL,**

**UserName VARCHAR(64) NULL,**

**UserKey VARCHAR(64) NULL,**

**UserPassPhrase VARCHAR(20),**

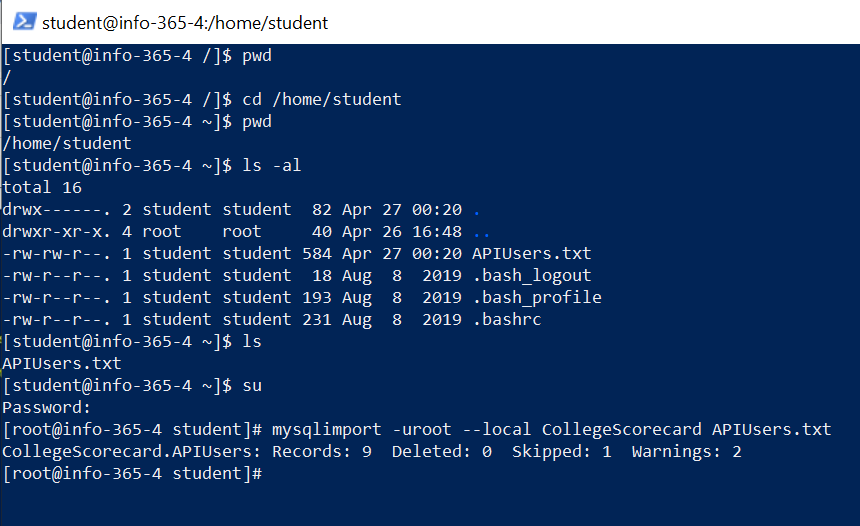
**PRIMARY KEY (ID)**

**)**

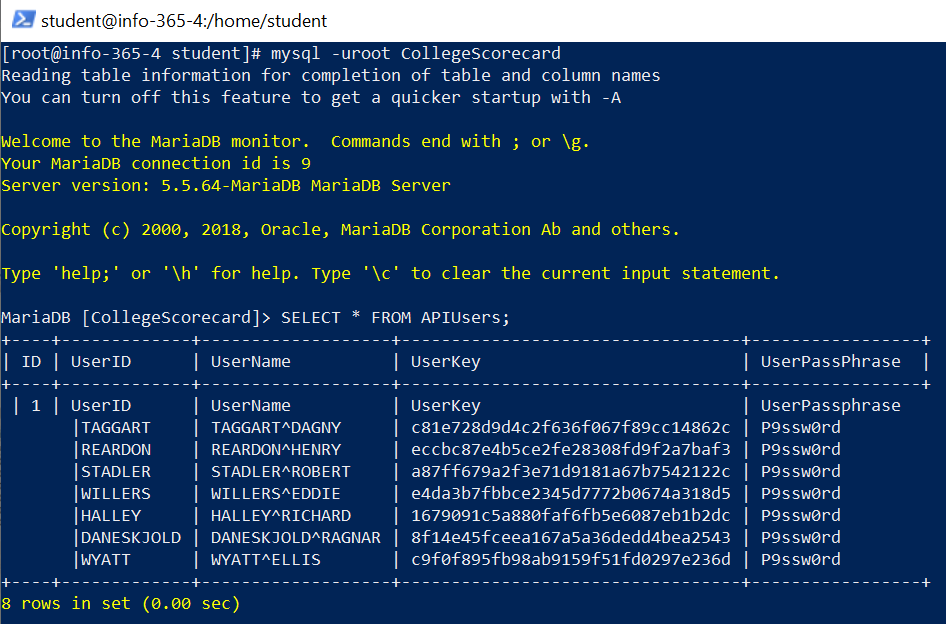
**ENGINE=InnoDB**

**COMMENT=’This is my user table’;**

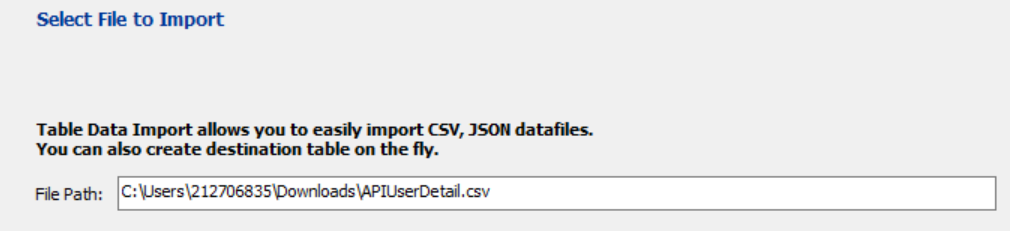
* 1. Import data to the APIUsers table using the mysqlimport utility:
     1. Copy the APIUsers.txt to your Linux server hosting MariaDB. Be sure to note the path to which you copied the file.
     2. Use SSH to login to your server, then use the “**cd**” command to change your working location to the directory in which the file you copied is located.
        1. For example, I copied my file into **/home/student**, so I would use the command “**cd /home/student**”.
        2. Verify your file is in your present directory, using the “**ls -l**” command. (*note, l = “L”, not “1”.*)
     3. Use the mysqlimport command to import the records from the text file you created and uploaded previously:



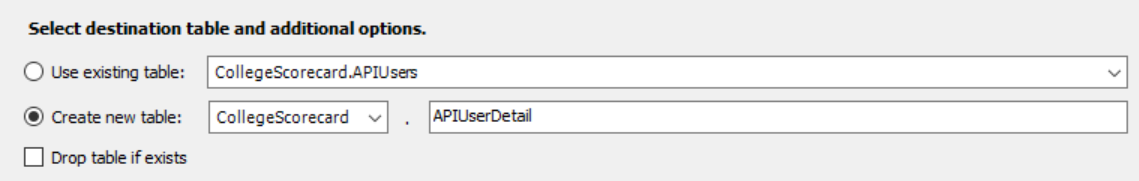
* + - 1. Notes: I first checked my current directory, which was root. I then used the “cd” command to change to the correct directory to which I copied my APIUsers.txt file. I verified the file exists using the “ls -al” command, then also used the “ls” command since it’s a little easier to read in the screen capture. I then ran the “mysqlimport” command, including the database name and filename to be used for the import.
      2. Some notes on mysqlimport:
         1. This command requires a text file, who’s name matches exactly with the name of a table in the database to which this data will be imported.
         2. The contents of the file MUST be a number of items on each line, separated by “tab” between each item, and the number of items must match the number of columns in the table to which they will be imported, and the columns must be in the same order.
         3. If your “root” user in MariaDB requires a password, you will need to add the **-p** option after the **-uroot** option.
    1. Login to the MariaDB server using the mysql command, and verify the table exists, and the data has been imported:



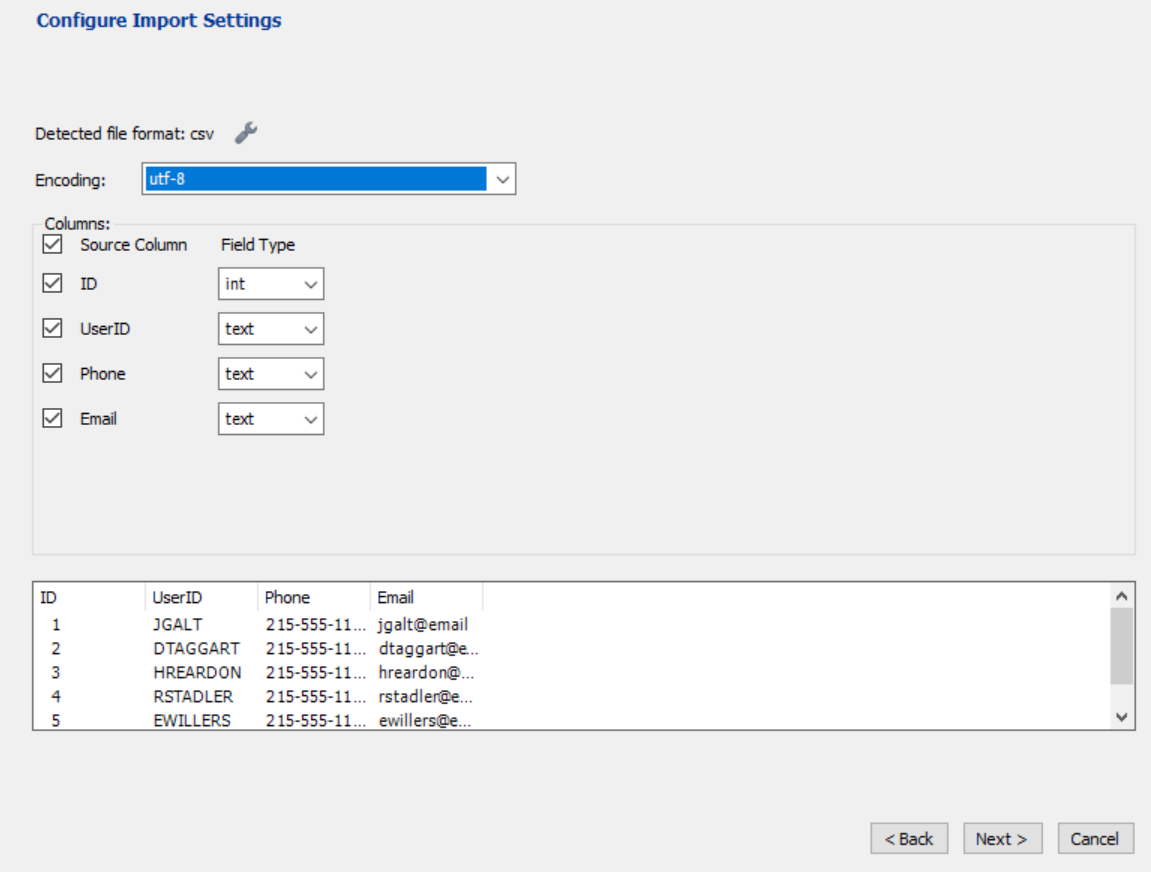
* 1. Import data to the APIUserDetail using MySQL Workbench’s “Table Import Wizard”
     1. Open MySQL Workbench, and connect to your database
        1. In some version of MySQL Workbench, you will need to click “Schemas” at the bottom of the left pane to see your database/tables.
     2. In the left Navigator pane, right click “Tables” (under the “CollegeScorecard” database).
     3. Select “Table Data Import Wizard”
     4. Select the CSV file we created earlier; mine is APIUserDatail.csv



* + 1. Select to create a new table (since we have not created this one yet), then ensure the database name is selected, and the table name is “APIUserDetail”.

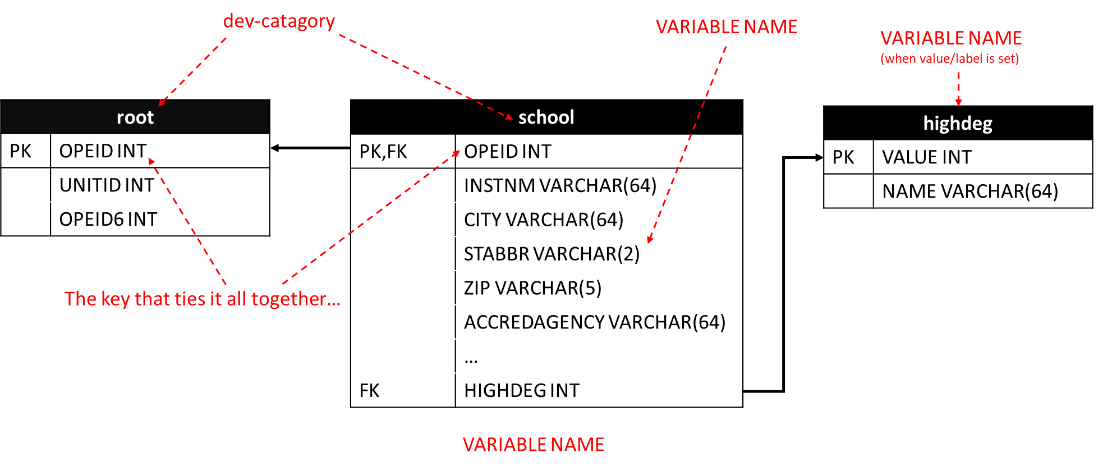


* + 1. The wizard should find the correct column names, datatypes, and set to UTF-8 (ASCII) for the encoding:

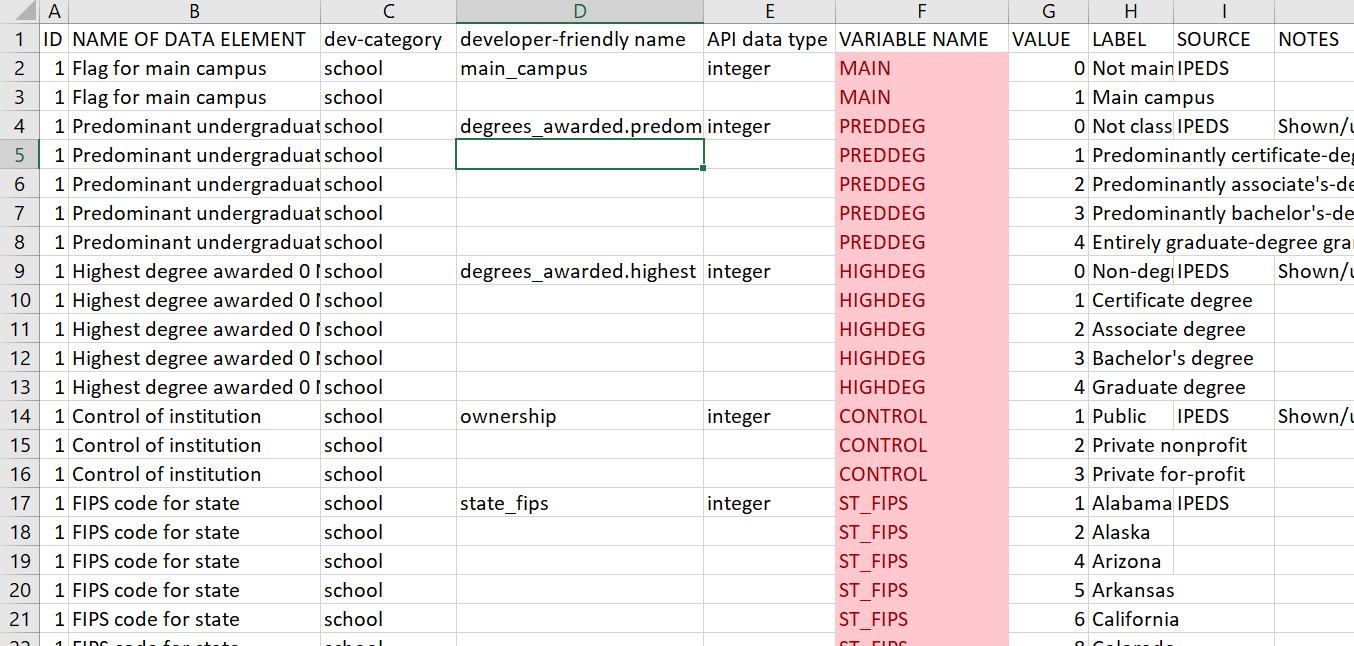


* + 1. Click “Next”, then “Next” on the following screen, which should attempt the import.
    2. After completing the import, refresh the table list, and verify the import worked.
  1. **Submit a screen capture image including both tables, one in MySQL Workbench, and the other with the mysql client on the server. Be sure to include your name in each screen capture image.**

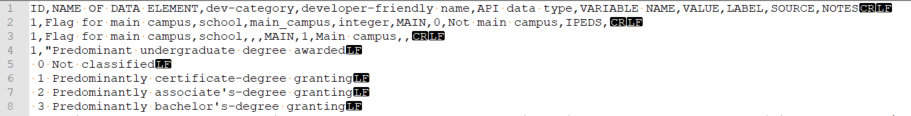
1. **Prepare the Data Files to create the Database Schema from the Data Dictionary:**
   1. To build the data dictionary, we can start by analyzing the Excel spreadsheet (**CollegeScorecard\_DataDictionary.xslt**).
   2. In class, we learned about [**normalization**](https://youtu.be/IcyfiUJX_0Y); the data provided by Data.Gov is a single table, that contains about 7,000 rows, and almost 2,000 columns. *We can assume that this is not in normal form*.
      1. We will need to analyze this data and create a normalized database schema.
      2. As you scroll through the columns and rows, look for columns that represents “lists”, or rows that contain repeating data, and think about ways to normalize this dataset.
      3. You may notice, that many of these columns are difficult to decode; for example, what is a UNITID? Or an OPEID? Or OPEID6?
      4. **Data.Gov provides a Data Dictionary file, which provides some additional information about this data, and will help us devise a relational, normalized database.**
   3. Note that there is sheet labeled “data\_dictionary”; this contains the information we will need to create our database and describes the data columns found in data CSV file.
      1. Note the **dev-category** column; one technique for identifying entities (which typically become tables in a physical database), is looking for categories of data; **we will use this column as the table name**.
      2. Note the column **VARIABLE NAME**; this matches the column names from the data file—**we will use this for the column names for each table**.
      3. Note the column **API data type**; we can use this to help determine the **datatype for each column**.
      4. The **developer-friendly-name** would probably make sense for use as the **comment** for each column, as it helps explain the purpose for each column.
      5. Note the **VALUE** and **LABEL** columns. This appears to indicate columns that have preset values; these values will need to be a separate table. Since the table defined in the dev-category table will contain the column that will reference the table with these key/value pairs, our foreign key will be in the table indicated in dev-category, while the table will the VARIABLE NAME.



* 1. First, we will need to create a data dictionary that describes all the tables on the right of the diagram above.
     1. Some of the columns in our dataset are defined to have preset values. This is a clue to some normalizes tables we will need… Any VARIABLE that referenced a “VALUE / LABEL” pair should be a foreign key to a table that contains those VALUE / LABEL pairs for that column.
     2. Use the column filter (each column should have a drop down arrow) and filter the VALUE column so only items with a value will be displayed by unchecking (blanks). You will now only have columns that represent these VALUE/LABEL pairs.
     3. Copy the contents that are displaying to a NEW instance of Excel
        1. You will have a new spreadsheet, with one tab, which contains a table of only the values we filtered.
     4. Note the “VARIABLE NAME” column… we will need to set all the blank values with a value, so this can be processed later by our script which will create the database.
        1. For example, rows 2 & 3 will show “MAIN” for the VARIABLE NAME. Likewise, rows 4-8 will have a value of PREDDEG, and so forth.
     5. Repeat the previous step for the NAME OF DATA ELEMENT column.
     6. Add a column to the left called ID and populate with a value (any integer is fine, the value is not important… as long as its not empty).

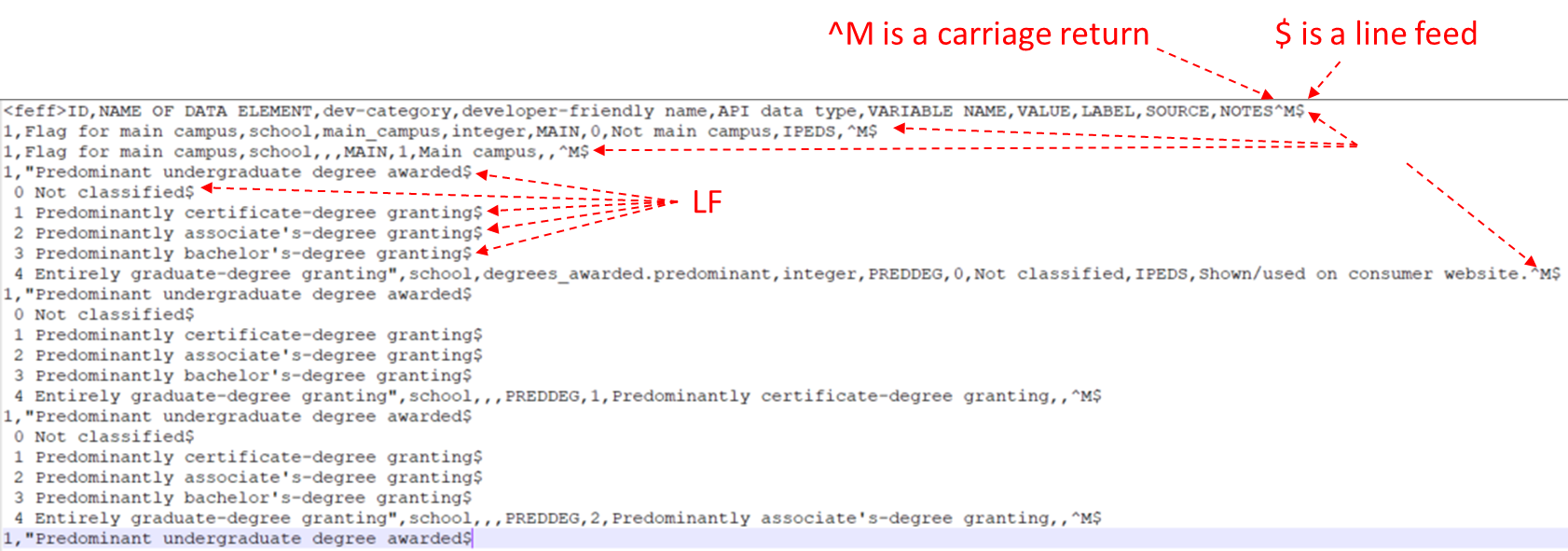


* + 1. Save this spreadsheet as a CSV (UTF-8) with the file name “CollegeScorecard\_OtherTables.csv”
       1. **Do not forget to clear the filter on the VALUE column which was set previously!**
    2. Open the CSV file you have created in Notepad++ (or another text editor you are familiar with), and it should look something like this:

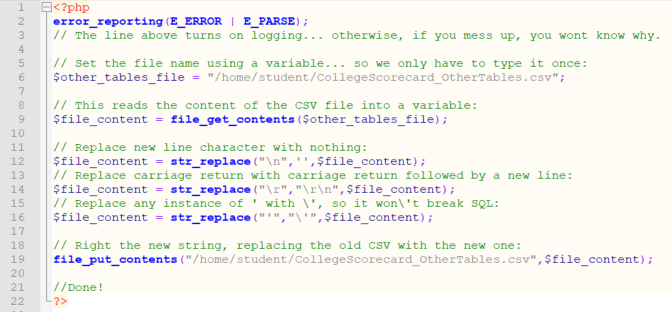


* 1. Next, we will need to save the dictionary file for use by the script. Using your original CollegeScorecardDataDictonary.xslx, ensuring you are on the “data\_dictionary” tab, save this sheet as “CollegeScorecard\_DataDictonary.csv” (ensure you use the CSV option, UTF-8).

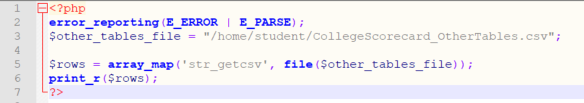
1. **Upload both CSV files you created, in addition to the original MERGED2016-17\_PP.csv to your Linux server hosting MariaDB.**
2. **Next, we will write a script to create the “Other Tables” using PHP. You will need to install PHP, as this is not typically installed with CentOS.** [**Follow the instructions in the appendix to install PHP**](#_Appendix_C:_Install)**.**
   1. It will also be helpful to know some basic PHP syntax. If you’ve previously taken courses in programming that covered syntax in Python, Perl, C, C++, the concept will be very familiar.
   2. Follow the tutorial in the appendix to [learn PHP](#_Appendix_D:_Some) syntax that will be useful for this (and future) assignments. If you already know the topics below, you can skip the tutorial:
      1. [Variables](#_PHP_Variables)
      2. [Arrays](#_PHP_Array)
      3. [String functions](#_PHP_String)
      4. [Conditions (If, Else)](#_PHP_-_The)
      5. [Loops](#_PHP_Loops) (For, While, Do, Foreach)
      6. [MySQLi](#_PHP_Connect_to) library to work with a database in PHP
   3. You should have already copied your dictionary files (both CSVs) to your server. My files are located in **/home/student** (if your files are in a different location, be sure to change the path in your script from my examples).
   4. First, we need to look at our CSV file in **VI**… there are some issues you will notice…
      1. Use the command [root@info-365-4 student]# **vi -b CollegeScorecard\_OtherTables.csv** to display the contents of the csv in VI.
      2. Once opened, type “**:set ffs=unix**”, then [**enter**]
      3. Next, type “**:set list**”, then [**enter**].
      4. Notice that most lines end with **^M$**. The **^M** is an ascii line feed, and the **$** is an ASCII carriage return. However, some lines end with just the **$** (line feed). These are lines where the source data included a line feed and will ruin our day when trying to parse them into building a schema. To fix this, we will use some PHP string functions. You will also notice that some lines contain an apostrophe; these will need to be prefixed with the SQL escape character “**\**” or our eventual SQL statements will fail.



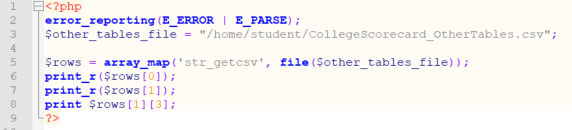
* + 1. To exit VI without saving the file, type [**Esc**] “**:q!**” [**Enter**]
    2. Create a PHP file called FixCSV.php, with the following contents (Note, comments were added to provide clarity; these are not required for your script to run):



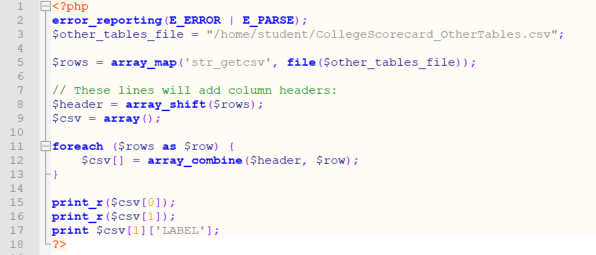
* + 1. Run this php script using the PHP command: [root@info-365-4 student]# **php FixCSV.php**
       1. Open your file with VI again and verify we have fixed the file.
       2. If you need to troubleshoot your script… be sure not to use the same copy of CollegeScorecard\_OtherTables.csv; the script might corrupt the file, you will want to start with a back-up copy each time you run the script. This is best practice for any of the flat files we will using with our scripts.
  1. Now, we are ready to build our script that creates the schema and insert statements:
     1. Create a file called CreateOtherTablesSchema.php, with the following content:



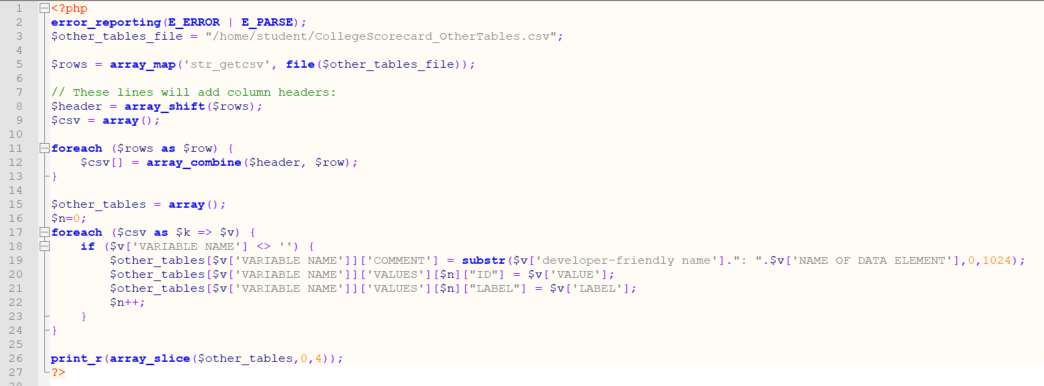
* + 1. To run this file: [root@info-365-4 student]# **php CreateOtherTables.php | less**
       1. Note: the “ | less “ at the end will output several lines at a time, so we can work with the file in the command line easier. Use your arrow keys to scroll up/down, then “**q**” to return to the command prompt.
          1. *The “ | “ character is a “pipe”; it is the typically above the Enter key on most keyboards, and requires “shift”.*
       2. **$rows** is an array variable, that is set by a function which converts the CSV into a multidimensional array. For example, edit your script with the following changes to line 6:



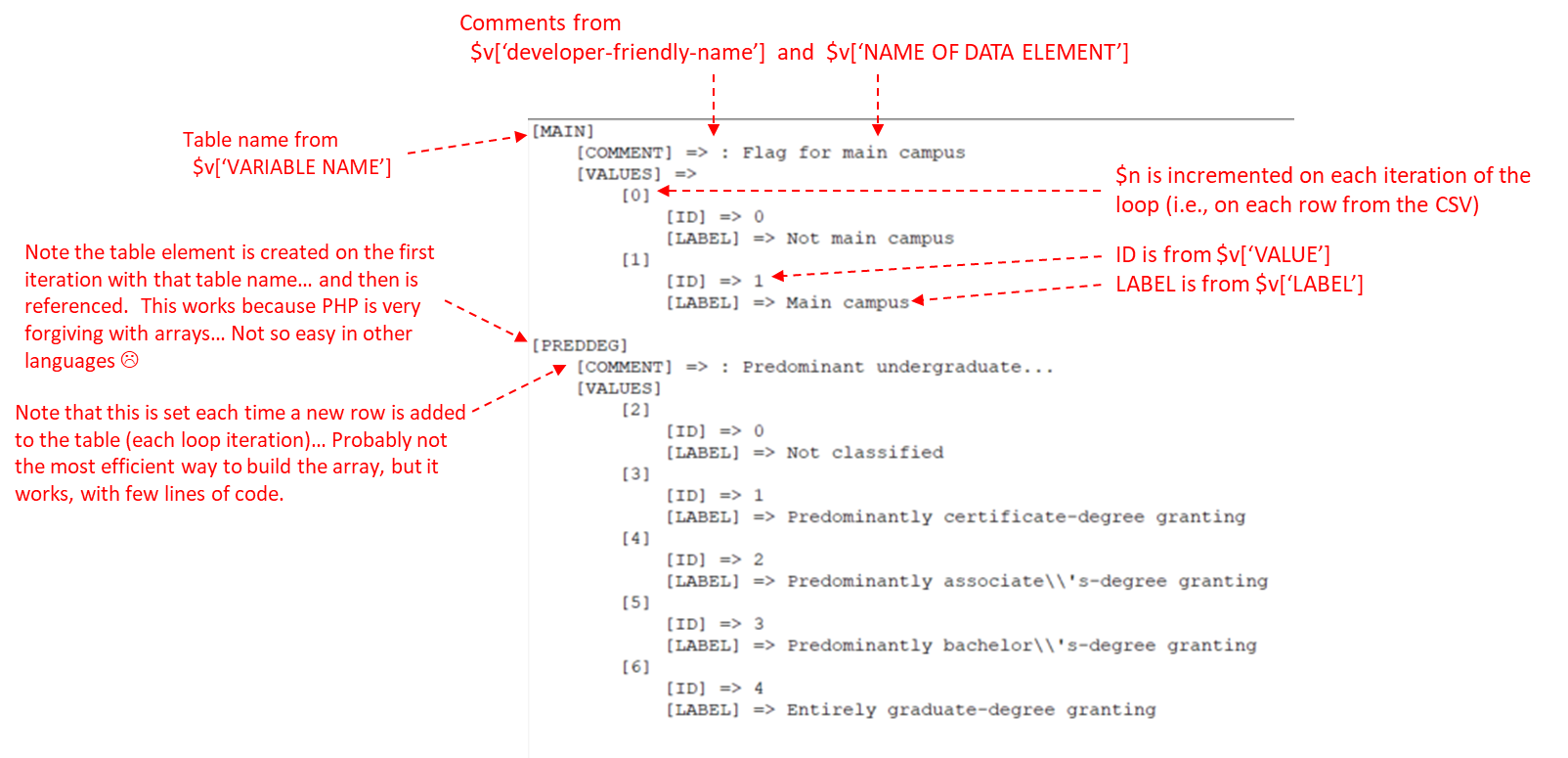
* + - 1. When running the script:
         1. line 6 outputs only the content of row 1 in the CSV (which are the column headers).
         2. Line 7 outputs only the content of row 2
         3. Line 8 only shows the 4th column of the 2nd row.
      2. Edit the script, adding lines 7-13, and editing lines 15-17 below. This will create a new array called $header, which contains the column headers, which will be combined with each row in the $rows array, so we will have an array we can reference by name rather than index for each column:



* + - * 1. Note, in line 17, we can now get a value from the column name; this will make it easier to work with the file.
      1. Edit the script, replacing from line 14-26:



* + - * 1. A few changes here… First, at line 15, we create an array called $other\_tables. This will contain a multidimensional array, which is populated by iterating through each row of the CSV (using $csv array we created earlier). The structure of the array will look like this:



* + - 1. Edit the script by adding lines 26-37:



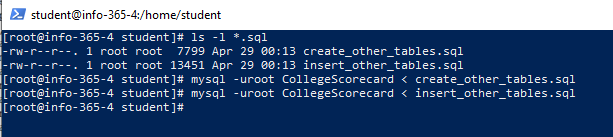
* + - * 1. This change adds a foreach loop, which will iterate through **$other\_tables**. The syntax **$table => $attributes** sets these variables on each iteration; **$table** will have the table name, while attributes will contain both the “**COMMENT**” and “**VALUES**” (values is another array, nested in this array).
        2. Lines 31-33 will append to **$create\_other\_tables** lines of SQL code based on the array we are iterating, to create each of these tables. Note that the first field is the table name (set to upper case using a string function) concatenated to **\_ID**, which will be the primary key.
        3. The last line (37) will write a file called **create\_other\_tables.sql**, which is a SQL script we should be able to run, which will create the tables.
        4. Open **create\_other\_tables.sql** and verify it has been populated with CREATE TABLE statements.
      1. Since we are here… we can easily add a few lines to the loop, which will be a nested loop to iterate to through the values each table will need and generate the INSERT statements for our database. Add the nested loop in lines 35-42 below (below line 33 in your previous version):



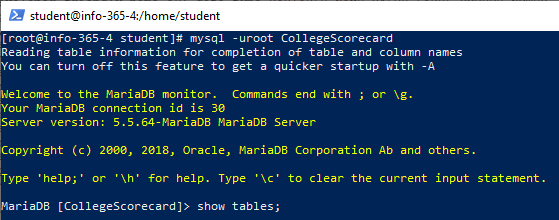
* + - * 1. This loop will iterate through the values for each table and generate insert statements. You will need to write the insert statements to a file by adding line 48 below:



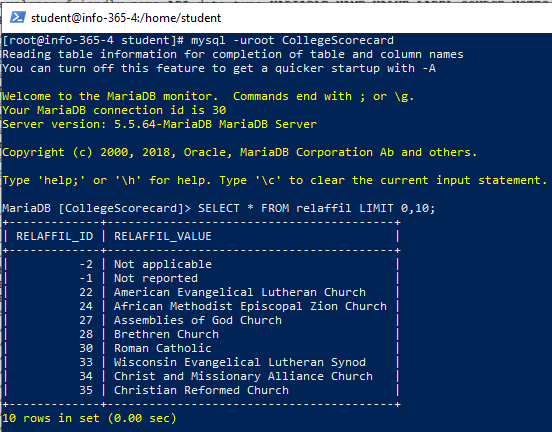
* + - 1. Use php to run this script one final time, and verify you have the two generated SQL files from lines 47 and 48, and verify they are populated.
    1. To run the SQL statements we generated, we will use the mysql client in Linux.
       1. In your SSH session, use the mysql command to run your SQL scripts:



* + - 1. Use the mysql command to verify the tables exist:

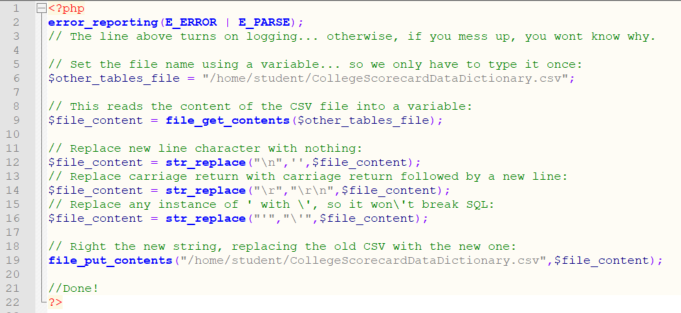


* + - 1. And are populated with data:

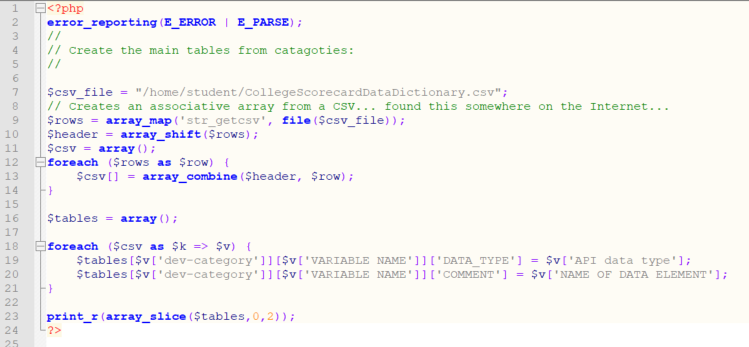


* 1. **Submit a screen capture showing the output of the “SHOW TABLES” command, which should show the supporting tables. Also include a screen capture showing the table that contains religious affiliation information. Be sure to include your name in all screen captures.**

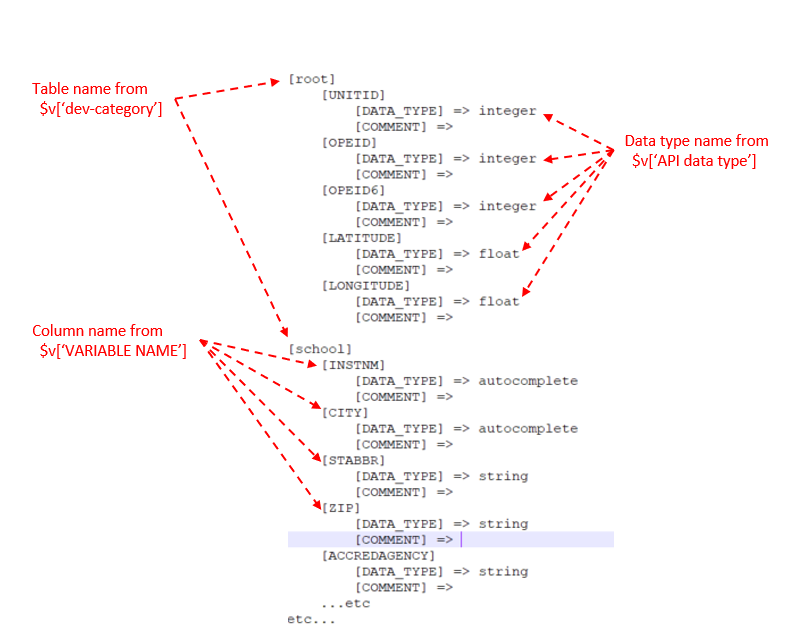
1. **Now we are ready to write a script that will create the remaining tables.**
   1. Modify the **CreateOtherTables.php**; edit lines 6 and 19 such the the script will proves **CollegeScorecardDataDictionary.csv** (rather than the previous CSV file):



* 1. Run the script using PHP on you Linux server.
  2. Create a PHP script (**CreateTables**.**php**) with the following contents:



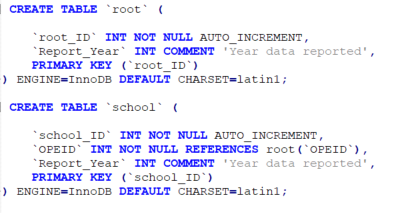
* 1. This is like the script we used to create “other tables”, but with a different structure for the array:



* 1. Modify the script by adding lines 23-46 below:



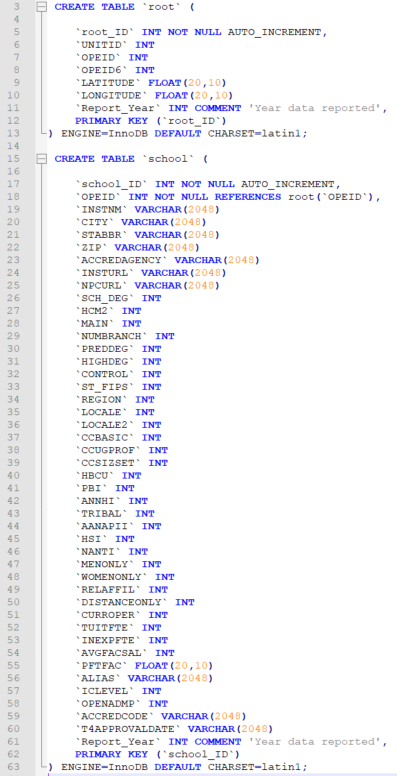
* + 1. The **create\_tables.sql** should generate something like this:



* 1. Modify the script so it will also add the columns to the create table statement; add lines 39-61 below after line 37 in the previous version:



* + 1. Your script will now output a file with fully formed CREATE TABLE statements, like this:

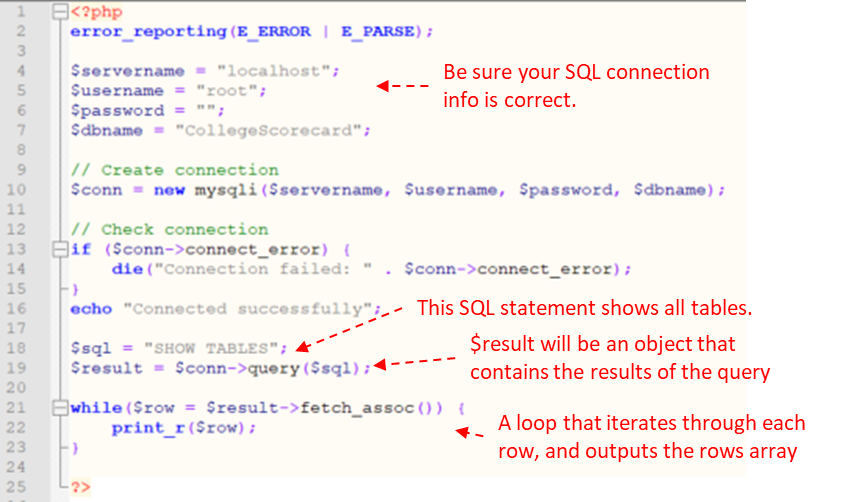


* 1. Use the mysql client utility to run the generated SQL script, to create the tables:

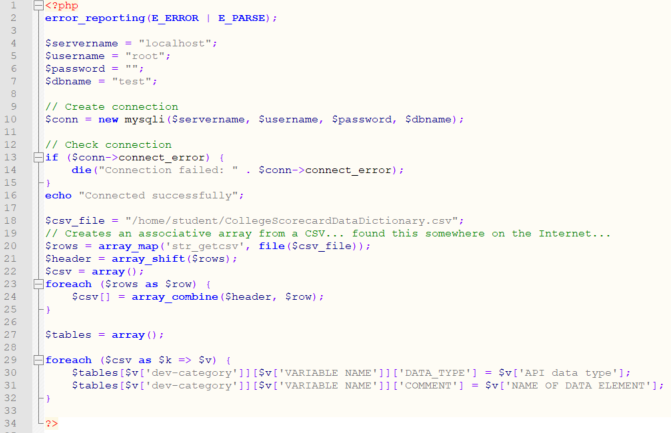


* 1. Use either MySQL Workbench or the mysql utility on your server and verify the tables have been created.

1. **To import the data, we will write a PHP script that parses MERGED2016\_17\_PP.csv (which contains all the data) and insert the records into the correct tables in the database.**
   1. Make sure the **MERGED2016\_17\_PP.csv** file is uploaded to your Linux server hosting MariaDB.
   2. Rather than generating insert statements, we will use PHP to directly insert the data. Let’s make sure PHP is able to work with MariaDB using the PHP MySQLi library:
      1. On your Linux server, ensuring you are at a root user prompt, use **yum** to install the php-mysqli package (**yum install mysqli**)
      2. Create a script called **Insert.php** with the following contents:



* + 1. After executing this script, you should see a list of tables displayed.
    2. Find the lines from your **CreateTables.php** script that creates the **$tables** array, and add to your **Inserts.php** (lines 18-32 below) and replace lines 18-23 from your previous **Inserts.php** version:



* + 1. Finally… We will add the code to read from the CSV and insert to the database:
       1. Since this is a much larger file, we will want to read one line at a time into memory (rather than the entire file):



* + - 1. Insert the following text (from above… but pasted here as text so you will not need to type it) into your **Insert.php** below line 32:

|  |
| --- |
| $csv\_file = "/home/student/MERGED2016\_17\_PP.csv";  $csv\_handle = fopen($csv\_file, "r");  $inserted\_count=0;  $failed\_count=0;  if ($csv\_handle) {  $n=0;  while (($line = fgets($csv\_handle)) !== false) {  if ($n > 8000) {break;}  if ($n==0) {  $csv\_header = explode(',',$line);  } else {  $csv\_columns = explode(',',$line);  // Make associated array:  $csv\_row = array();  for ($c=1;$c < count($csv\_header);$c++) {  $csv\_row[trim($csv\_header[$c])] = trim($csv\_columns[$c]);  }  foreach ($tables as $table => $columns) {  if (trim($table) <> '' && trim($table) <> 'completion') {  $inserts = "\n\nINSERT INTO `".$table."`";  // Column/Value List:  $columns\_list = "";  $values\_list = "";  foreach ($columns as $column => $attributes) {  if (trim($column) <> '' && trim($column) <> 'OPEID') {  $columns\_list = $columns\_list."`".$column."`,";  if ($csv\_row[$column] == 'NULL' || $csv\_row[$column] == 'PrivacySuppressed' || $csv\_row[$column] == '') {  $values\_list = $values\_list."NULL,";  } elseif ($attributes["DATA\_TYPE"] == 'integer' || $attributes["DATA\_TYPE"] == 'float') {  if (is\_numeric($csv\_row[$column])) {  $values\_list = $values\_list.$csv\_row[$column].",";  } else {  $values\_list = $values\_list."NULL,";  }  } else {  $values\_list = $values\_list."'".mysqli\_real\_escape\_string($conn,$csv\_row[$column])."',";  }  }  }  $columns\_list = $columns\_list." `OPEID`";  $values\_list = $values\_list.$csv\_row['OPEID'];  $inserts = "\n\nINSERT INTO `".$table."` (".$columns\_list.") VALUES (".$values\_list.");";  if ($conn->query($inserts) === TRUE) {  $inserted\_count++;  } else {  echo "\nOPEID".$csv\_row['OPEID']." FAILED for table ".$table."\n\n\t" . $conn->error;  echo "\n\n".$inserts;  $failed\_count++;  }  }  }  }  $n++;  }  fclose($csv\_handle);  echo "\n\nSuccessfully Inserted: ".$inserted\_count;  echo "\n\nFailed Inserted: ".$failed\_count;  } else {  echo "\n\nERROR!";  } |

* + 1. Run your **insert.php** script and note the output.
       1. Grab some coffee… It might take a while for the script to run
    2. Verify you have records inserted in your database, using the following SQL script (either in MySQL Workbench or using the mysql utility):

**SELECT COUNT(\*) FROM root;**

**SELECT COUNT(\*) FROM school;**

**SELECT COUNT(\*) FROM admissions;**

**SELECT COUNT(\*) FROM academics;**

**SELECT COUNT(\*) FROM student;**

**SELECT COUNT(\*) FROM cost;**

**SELECT COUNT(\*) FROM aid;**

**SELECT COUNT(\*) FROM completion;**

**SELECT COUNT(\*) FROM repayment;**

**SELECT COUNT(\*) FROM earnings;**

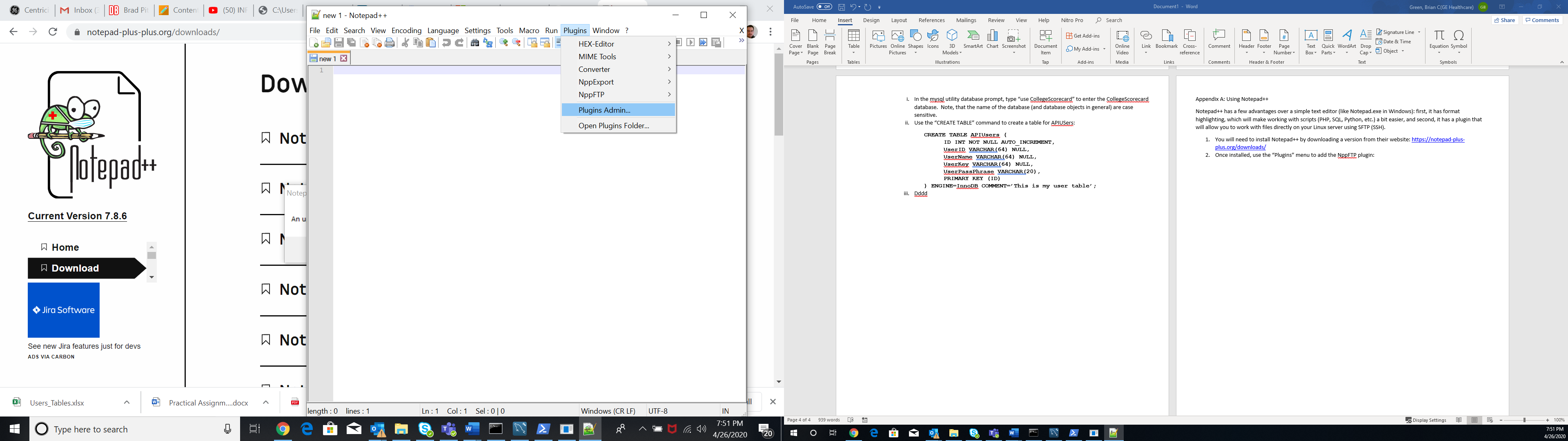
* + 1. Verify each table’s count is approximately 7,000 records.

1. **Using MySQL Workbench, generate a database diagram**
   1. Open MySQL Workbench, and connect to your database.
   2. In the Database menu, select “Reverse Engineer” (or just press CTRL-R)
   3. From the “Stored Connection” drop down, select your server.
   4. Click “Next”, and ensure the tool was able to connect to the database, retrieve the schema, and passed the common issues test.
   5. Click “Next”, and select your CollegeScorecard database.
   6. Click “Next a few more times, until you get to the screen with the “Execute >” button; ensure “Place imported objects on a diagram” is selected, then click “Execute >”
   7. Click “Next”, then “Finish”
   8. **Submit a PNG file of your completed EER diagram.**
2. **Submit a screen capture showing the results of the following two queries:**
   1. **All institution names with their religious affiliation.**
   2. **Information about Drexel University.**

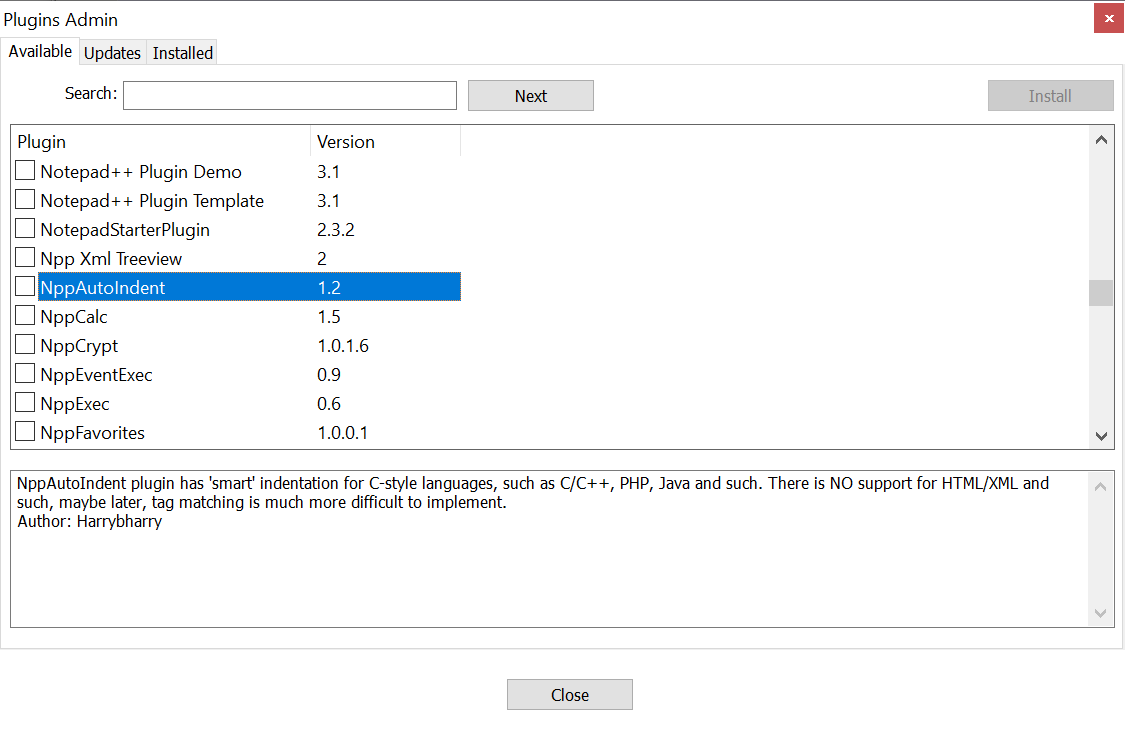
## Appendix A: Using Notepad++

Notepad++ has a few advantages over a simple text editor (like Notepad.exe in Windows): first, it has format highlighting, which will make working with scripts (PHP, SQL, Python, etc.) a bit easier, and second, it has a plugin that will allow you to work with files directly on your Linux server using SFTP (SSH).

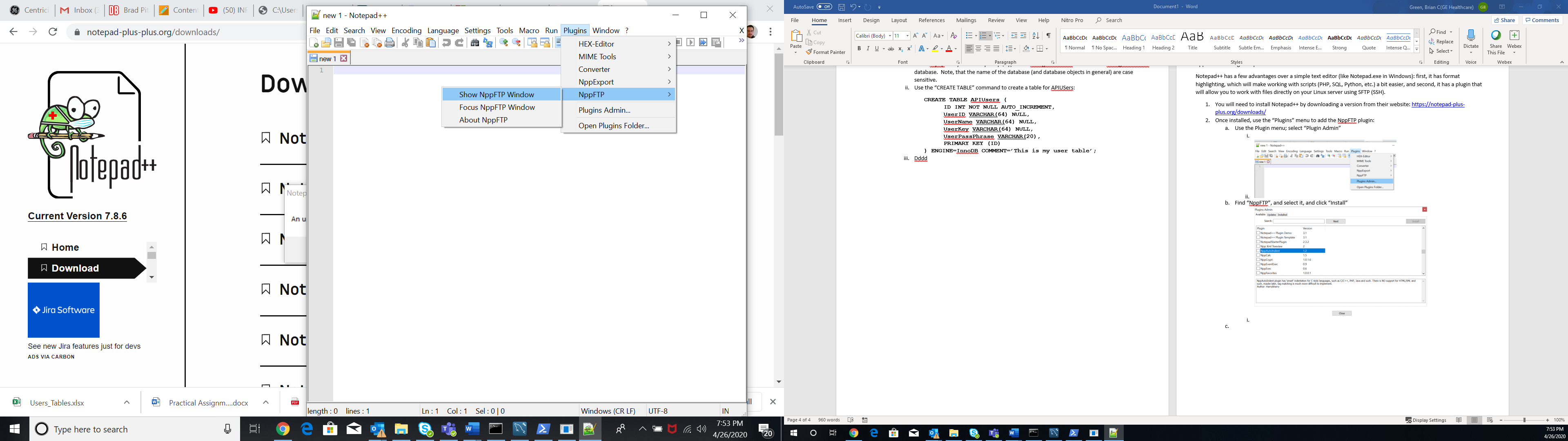
1. You will need to install Notepad++ by downloading a version from their website: <https://notepad-plus-plus.org/downloads/>
2. Once installed, use the “Plugins” menu to add the NppFTP plugin:
   1. Use the Plugin menu; select “Plugin Admin”



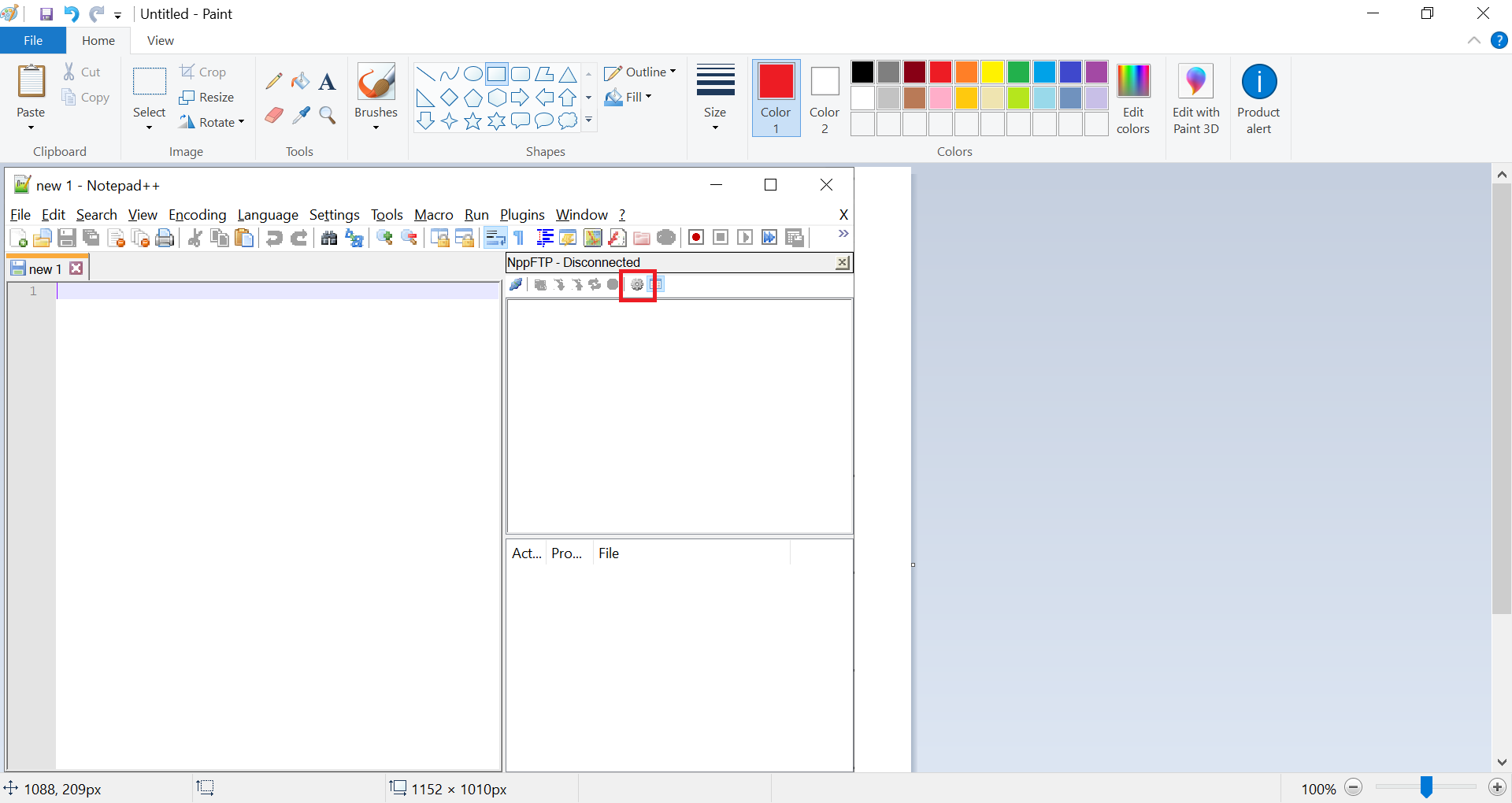
* 1. Find “NppFTP”, and select it, and click “Install”



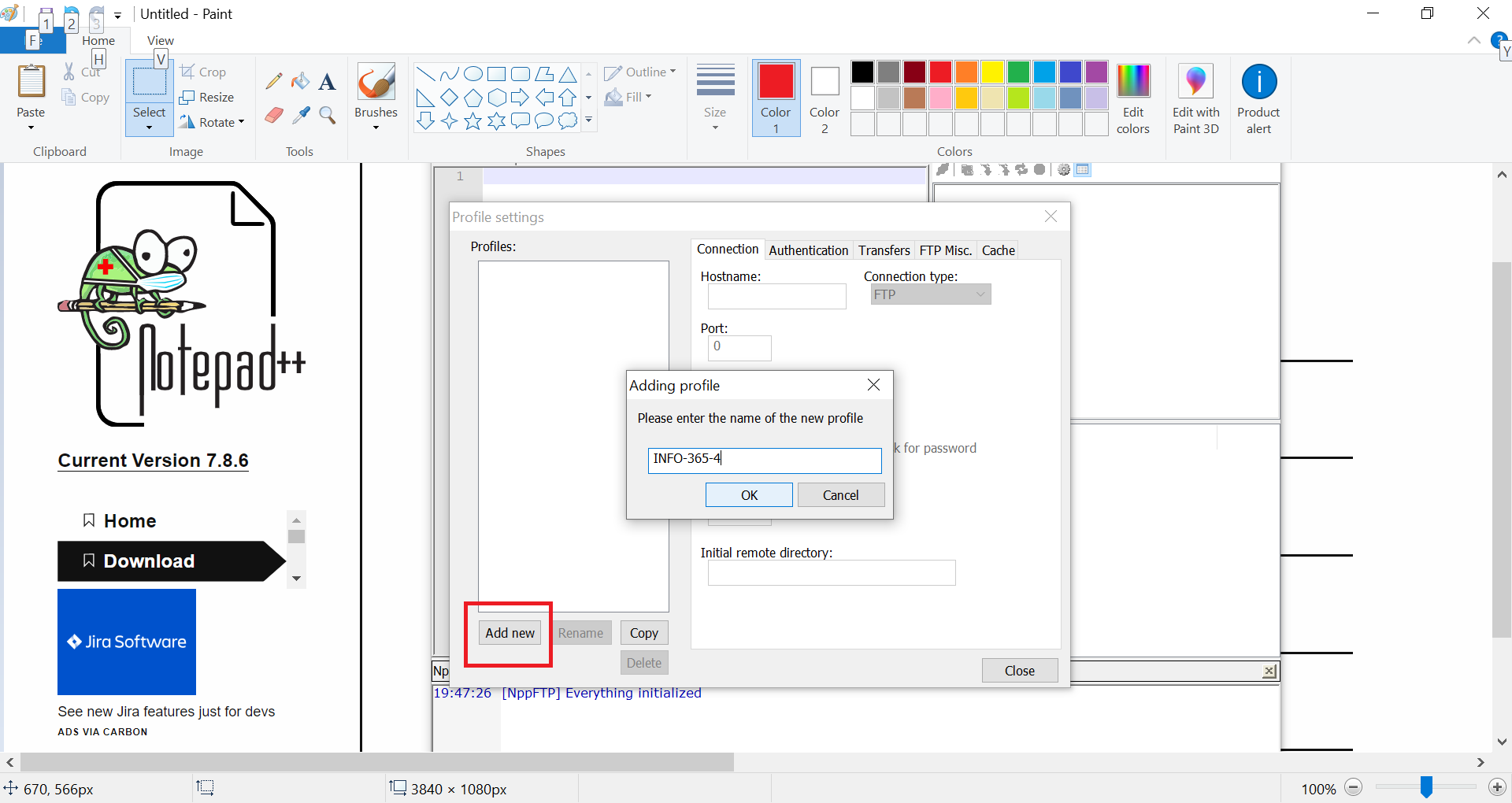
* 1. In the Plugins menu, select NppFTP, and “Show NppFTP Window”



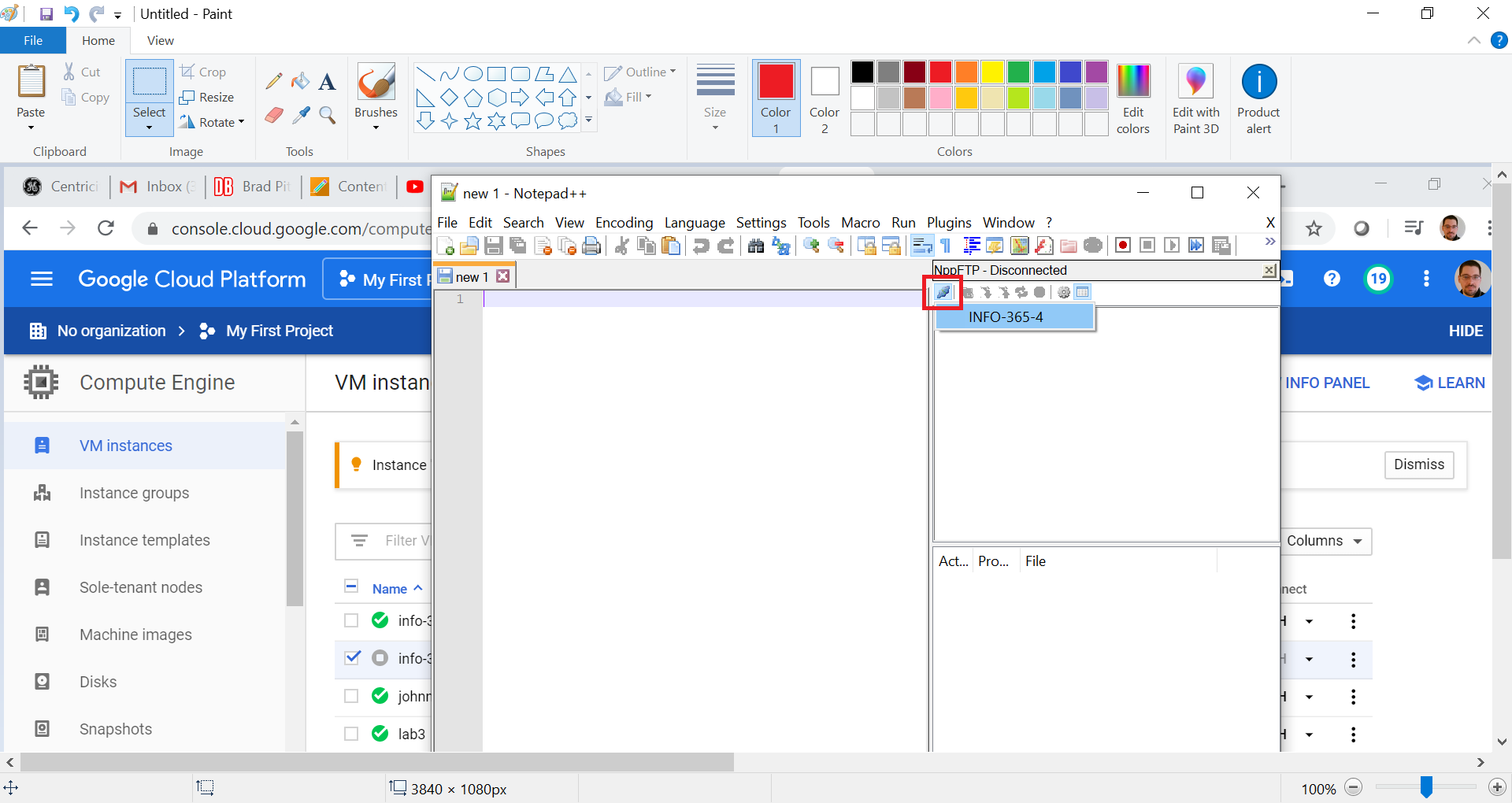
* 1. On the right pane that appears, click the settings icon, then select “Profile Settings”:



* 1. On the left, click “Add New”, and give the profile a name, then click OK.



* 1. Complete the fields for your Linux VM that is hosting MariaDB:
     1. **Hostname**: this will be the **External IP** address of your VM
     2. **Connection Type**: SFTP
     3. **Port**: 22
     4. **Username**: the non-root Linux username you created in PA-1
     5. **Password**: the password for your non-root username for Linux
     6. **Click “Close”** (Note, there is no save button… closing will save the form)
  2. Click the “connect” button, which looks like a set of plugs, in the upper left of the NppFTP plugin pane, and select the profile you created:



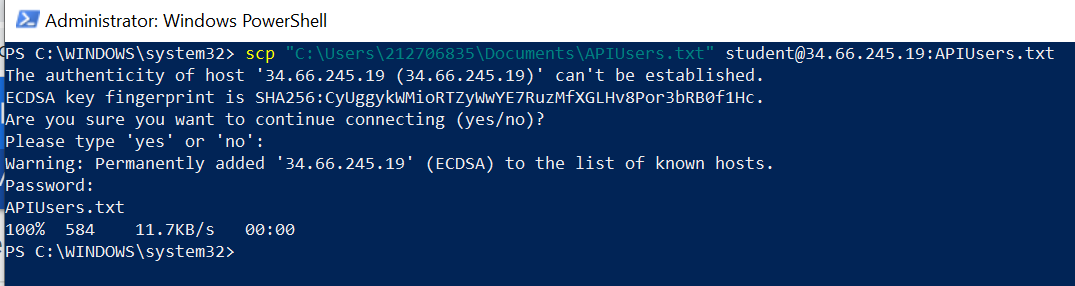
* 1. You can now see files in your user’s home directory
     1. You can right click on a directory to create a file, and double click to load in NppFTP. Anything you save that was opened from your server, will automatically save back to your server.

## Appendix B: Using the SCP command

SCP (Secure Copy) allows you to copy files from your computer to your Linux server directly, using SSH. This command works in both MAC (from the shell prompt) and Windows (using PowerShell).

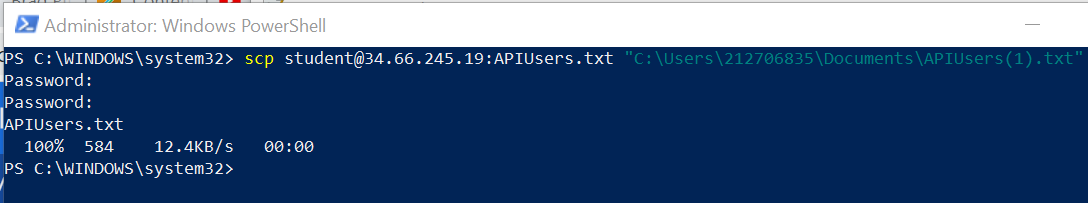
To use this command, it is like the **cp** command, except the source and destination can reference locations on a network.

For example, to copy a local file called APIUsers.txt located in **C:\Users\212706835\Documents** on my local computer to a remote server with the External IP of 34.66.245.19 that has a user called “Student”, I would use the following command:



Note: The first line contains the command; I used double quotes around the full file path/name, as this would be required if there are special characters or spaces in the path or file name. The second though fifth lines are accepting the SSH certificate, for which you will be prompted if this is the first time you have used the remote location. The sixth line prompts me for my Linux user’s password, then we see the file has been copied.

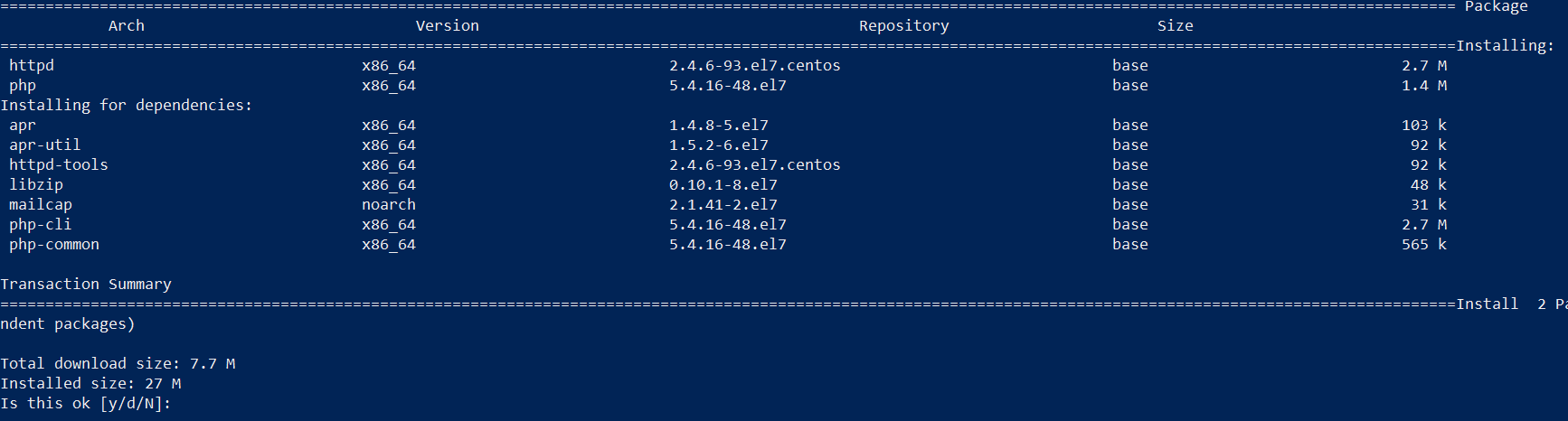
Likewise, I could copy this file from my remote server to my local machine:



## Appendix C: Install PHP on CentOS

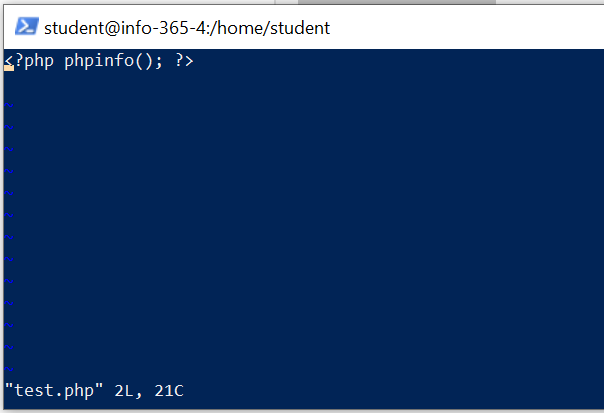
To install PHP, we will use a package manager, similar to how MariaDB was installed (using the yum command). Be sure you have root permissions (either use the “su” command to get a root prompt, or preface each command with “sudo” while logged in with a user in the sodoers group).

[root@info-365-4 student]# **yum install httpd php**



Test your PHP install:

1. Use vim to create a file called “test.php”: [root@info-365-4 student]# **vim test.php**
   1. Type the letter “i” to enter INSERT mode
   2. Type the line as shown below:



* 1. Press “Esc”, then type “:wq!”
  2. Type the command: [root@info-365-4 student]# **php test.php**
     1. If you have an error message, something is wrong.
     2. You should see a wall of text, describing your PHP install.

## Appendix D: Some PHP Basics

PHP is an easy scripting language to learn. Although typically used for web development, it makes for a decent fully capable scripting language when working with databases (it includes native drivers for most database engines, including MariaDB).

The following excerpts are from <https://www.w3schools.com/php>; I have adapted them for use as a scripting (rather than web development) language.

A PHP script starts with <?php and ends with ?>:

<?php  
// PHP code goes here  
?>

The default file extension for PHP files is ".php".

In PHP, NO keywords (e.g. if, else, while, echo, etc.), classes, functions, and user-defined functions are case-sensitive.

In the example below, all three echo statements below are equal and legal:

<?php  
ECHO "Hello World!;  
echo "Hello World!;  
EcHo "Hello World!;  
?>

Look at the example below; only the first statement will display the value of the $color variable! This is because $color, $COLOR, and $coLOR are treated as three different variables:

<?php  
$color = "red";  
echo "My car is " . $color ;  
echo "My house is " . $COLOR ;  
echo "My boat is " . $coLOR ;  
?>

### Comments in PHP

A comment in PHP code is a line that is not executed as a part of the program. Its only purpose is to be read by someone who is looking at the code.

Comments can be used to:

* Let others understand your code
* Remind yourself of what you did - Most programmers have experienced coming back to their own work a year or two later and having to re-figure out what they did. Comments can remind you of what you were thinking when you wrote the code

PHP supports several ways of commenting:

Syntax for single-line comments:

<?php  
// This is a single-line comment  
  
# This is also a single-line comment  
?>

Syntax for multiple-line comments:

<?php  
/\*  
This is a multiple-lines comment block  
that spans over multiple  
lines  
\*/  
?>

### Creating (Declaring) PHP Variables

In PHP, a variable starts with the $ sign, followed by the name of the variable:

<?php  
$txt = "Hello world!";  
$x = 5;  
$y = 10.5;  
?>

After the execution of the statements above, the variable $txt will hold the value Hello world!, the variable $x will hold the value 5, and the variable $y will hold the value 10.5.

Note: When you assign a text value to a variable, put quotes around the value.

Note: Unlike other programming languages, PHP has no command for declaring a variable. It is created the moment you first assign a value to it.

Think of variables as containers for storing data.

### PHP Variables

A variable can have a short name (like x and y) or a more descriptive name (age, carname, total\_volume).

Rules for PHP variables:

* A variable starts with the $ sign, followed by the name of the variable
* A variable name must start with a letter or the underscore character
* A variable name cannot start with a number
* A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and \_ )
* Variable names are case-sensitive ($age and $AGE are two different variables)

Remember that PHP variable names are case-sensitive!

### Output Variables

The PHP echo statement is often used to output data to the screen.

The following example will show how to output text and a variable:

<?php  
$txt = "W3Schools.com";  
echo "I love $txt!";  
?>

The following example will produce the same output as the example above:

<?php  
$txt = "W3Schools.com";  
echo "I love " . $txt . "!";  
?>

The following example will output the sum of two variables:

<?php  
$x = 5;  
$y = 4;  
echo $x + $y;  
?>

### PHP is a Loosely Typed Language

In the example above, notice that we did not have to tell PHP which data type the variable is.

PHP automatically associates a data type to the variable, depending on its value. Since the data types are not set in a strict sense, you can do things like adding a string to an integer without causing an error.

In PHP 7, type declarations were added. This gives an option to specify the data type expected when declaring a function, and by enabling the strict requirement, it will throw a "Fatal Error" on a type mismatch.

You will learn more about strict and non-strict requirements, and data type declarations in the [PHP Functions](https://www.w3schools.com/php/php_functions.asp) chapter.

### PHP Variables Scope

In PHP, variables can be declared anywhere in the script.

The scope of a variable is the part of the script where the variable can be referenced/used.

PHP has three different variable scopes:

* local
* global
* static

### Global and Local Scope

A variable declared outside a function has a GLOBAL SCOPE and can only be accessed outside a function:

<?php  
$x = 5; // global scope  
  
function myTest() {  
    // using x inside this function will generate an error  
    echo "Variable x inside function is: $x";  
}  
myTest();  
  
echo "Variable x outside function is: $x";  
?>

A variable declared within a function has a LOCAL SCOPE and can only be accessed within that function:

<?php  
function myTest() {  
    $x = 5; // local scope  
    echo "Variable x inside function is: $x";  
}  
myTest();  
  
// using x outside the function will generate an error  
echo "Variable x outside function is: $x";  
?>

### PHP The global Keyword

The global keyword is used to access a global variable from within a function.

To do this, use the global keyword before the variables (inside the function):

<?php  
$x = 5;  
$y = 10;  
  
function myTest() {  
    global $x, $y;  
    $y = $x + $y;  
}  
  
myTest();  
echo $y; // outputs 15  
?>

PHP also stores all global variables in an array called $GLOBALS[*index*]. The *index* holds the name of the variable. This array is also accessible from within functions and can be used to update global variables directly.

The example above can be rewritten like this:

<?php  
$x = 5;  
$y = 10;  
  
function myTest() {  
    $GLOBALS['y'] = $GLOBALS['x'] + $GLOBALS['y'];  
}  
  
myTest();  
echo $y; // outputs 15  
?>

### PHP The static Keyword

Normally, when a function is completed/executed, all of its variables are deleted. However, sometimes we want a local variable NOT to be deleted. We need it for a further job.

To do this, use the static keyword when you first declare the variable:

<?php  
function myTest() {  
    static $x = 0;  
    echo $x;  
    $x++;  
}  
  
myTest();  
myTest();  
myTest();  
?>

Then, each time the function is called, that variable will still have the information it contained from the last time the function was called.

Note: The variable is still local to the function.

### PHP echo and print Statements

echo and print are more or less the same. They are both used to output data to the screen.

The differences are small: echo has no return value while print has a return value of 1 so it can be used in expressions. echo can take multiple parameters (although such usage is rare) while print can take one argument. echo is marginally faster than print.

### The PHP echo Statement

The echo statement can be used with or without parentheses: echo or echo().

### ****Display Text****

The following example shows how to output text with the echo command (notice that the text can contain HTML markup):

<?php  
echo "PHP is Fun!";  
echo "Hello world!;  
echo "I'm about to learn PHP!;  
echo "This ", "string ", "was ", "made ", "with multiple parameters.";  
?>

**Display Variables**

The following example shows how to output text and variables with the echo statement:

<?php  
$txt1 = "Learn PHP";  
$txt2 = "W3Schools.com";  
$x = 5;  
$y = 4;  
  
echo $txt1;  
echo "Study PHP at " . $txt2 ;  
echo $x + $y;  
?>

### The PHP print Statement

The print statement can be used with or without parentheses: print or print().

### **Display **Text****

The following example shows how to output text with the print command (notice that the text can contain HTML markup):

<?php  
print "PHP is Fun!";  
print "Hello world!;  
print "I'm about to learn PHP!";  
?>

**Display Variables**

The following example shows how to output text and variables with the print statement:

<?php  
$txt1 = "Learn PHP";  
$txt2 = "W3Schools.com";  
$x = 5;  
$y = 4;  
  
print $txt1  
print "Study PHP at " . $txt2 ;  
print $x + $y;  
?>

### PHP Data Types

Variables can store data of different types, and different data types can do different things.

PHP supports the following data types:

* String
* Integer
* Float (floating point numbers - also called double)
* Boolean
* Array
* Object
* NULL
* Resource

### PHP String

A string is a sequence of characters, like "Hello world!".

A string can be any text inside quotes. You can use single or double quotes:

<?php  
$x = "Hello world!";  
$y = 'Hello world!';  
  
echo $x;  
echo ";  
echo $y;  
?>

### PHP Integer

An integer data type is a non-decimal number between -2,147,483,648 and 2,147,483,647.

Rules for integers:

* An integer must have at least one digit
* An integer must not have a decimal point
* An integer can be either positive or negative
* Integers can be specified in: decimal (base 10), hexadecimal (base 16), octal (base 8), or binary (base 2) notation

In the following example $x is an integer. The PHP var\_dump() function returns the data type and value:

<?php  
$x = 5985;  
var\_dump($x);  
?>

### PHP Float

A float (floating point number) is a number with a decimal point or a number in exponential form.

In the following example $x is a float. The PHP var\_dump() function returns the data type and value:

<?php  
$x = 10.365;  
var\_dump($x);  
?>

### PHP Boolean

A Boolean represents two possible states: TRUE or FALSE.

$x = true;  
$y = false;

Booleans are often used in conditional testing. You will learn more about conditional testing in a later chapter of this tutorial.

### PHP Array

An array stores multiple values in one single variable.

In the following example $cars is an array. The PHP var\_dump() function returns the data type and value:

<?php  
$cars = array("Volvo","BMW","Toyota");  
var\_dump($cars);  
?>

### PHP Resource

The special resource type is not an actual data type. It is the storing of a reference to functions and resources external to PHP.

A common example of using the resource data type is a database call.

We will not talk about the resource type here, since it is an advanced topic.

### PHP String Functions

### strlen() - Return the Length of a String

The PHP strlen() function returns the length of a string.

<?php  
echo strlen("Hello world!"); // outputs 12  
?>

### Complete PHP String Reference

For a complete reference of all string functions, go to our complete [PHP String Reference](https://www.w3schools.com/php/php_ref_string.asp).

The PHP string reference contains description and example of use, for each function!

### PHP Integers

An integer is a number without any decimal part.

2, 256, -256, 10358, -179567 are all integers. While 7.56, 10.0, 150.67 are floats.

So, an integer data type is a non-decimal number between -2147483648 and 2147483647. A value greater (or lower) than this, will be stored as float, because it exceeds the limit of an integer.

Another important thing to know is that even if 4 \* 2.5 is 10, the result is stored as float, because one of the operands is a float (2.5).

Here are some rules for integers:

* An integer must have at least one digit
* An integer must not have a decimal point
* An integer can be either positive or negative
* Integers can be specified in three formats: decimal (10-based), hexadecimal (16-based - prefixed with 0x) or octal (8-based - prefixed with 0)

PHP has the following functions to check if the type of a variable is integer:

* is\_int()
* is\_integer() - alias of is\_int()
* is\_long() - alias of is\_int()

<?php  
$x = 5985;  
var\_dump(is\_int($x));  
  
$x = 59.85;  
var\_dump(is\_int($x));  
?>

### PHP Floats

A float is a number with a decimal point or a number in exponential form.

2.0, 256.4, 10.358, 7.64E+5, 5.56E-5 are all floats.

The float data type can commonly store a value up to 1.7976931348623E+308 (platform dependent), and have a maximum precision of 14 digits.

PHP has the following functions to check if the type of a variable is float:

* is\_float()
* is\_double() - alias of is\_float()

<?php  
$x = 10.365;  
var\_dump(is\_float($x));  
?>

### PHP Conditional Statements

Very often when you write code, you want to perform different actions for different conditions. You can use conditional statements in your code to do this.

In PHP we have the following conditional statements:

* if statement - executes some code if one condition is true
* if...else statement - executes some code if a condition is true and another code if that condition is false
* if...elseif...else statement - executes different codes for more than two conditions
* switch statement - selects one of many blocks of code to be executed

### PHP - The if Statement

The if statement executes some code if one condition is true.

Syntax

if (*condition*) { *code to be executed if condition is true*;  
}

<?php  
$t = date("H");  
  
if ($t < "20") {  
    echo "Have a good day!";  
}  
?>

### PHP - The if...else Statement

The if...else statement executes some code if a condition is true and another code if that condition is false.

### Syntax

if (*condition*) {  
    *code to be executed if condition is true;*  
} else {  
  *code to be executed if condition is false;*}

<?php  
$t = date("H");  
  
if ($t < "20") {  
    echo "Have a good day!";  
} else {  
    echo "Have a good night!";  
}  
?>

### PHP - The if...elseif...else Statement

The if...elseif...else statement executes different codes for more than two conditions.

Syntax

if (*condition*) {  
    *code to be executed if this condition is true;*} elseif (*condition*) {  
  *code to be executed if first condition is false and this condition is true;*} else {  
    *code to be executed if all conditions are false;*}

<?php  
$t = date("H");  
  
if ($t < "10") {  
    echo "Have a good morning!";  
} elseif ($t < "20") {  
    echo "Have a good day!";  
} else {  
    echo "Have a good night!";  
}  
?>

### PHP Loops

Often when you write code, you want the same block of code to run over and over again a certain number of times. So, instead of adding several almost equal code-lines in a script, we can use loops.

Loops are used to execute the same block of code again and again, as long as a certain condition is true.

In PHP, we have the following loop types:

* while - loops through a block of code as long as the specified condition is true
* do...while - loops through a block of code once, and then repeats the loop as long as the specified condition is true
* for - loops through a block of code a specified number of times
* foreach - loops through a block of code for each element in an array

The following chapters will explain and give examples of each loop type.

### The PHP while Loop

The while loop executes a block of code as long as the specified condition is true.

Syntax

while (*condition is true*) {  
*code to be executed*;  
}

Examples

The example below displays the numbers from 1 to 5:

<?php  
$x = 1;  
  
while($x <= 5) {  
    echo "The number is: $x ;  
    $x++;  
}  
?>

Example Explained:

* $x = 1; - Initialize the loop counter ($x), and set the start value to 1
* $x <= 5 - Continue the loop as long as $x is less than or equal to 5
* $x++; - Increase the loop counter value by 1 for each iteration

This example counts to 100 by tens:

<?php  
$x = 0;  
  
while($x <= 100) {  
    echo "The number is: $x ;  
    $x+=10;  
}  
?>

Example Explained

* $x = 0; - Initialize the loop counter ($x), and set the start value to 0
* $x <= 100 - Continue the loop as long as $x is less than or equal to 100
* $x+=10; - Increase the loop counter value by 10 for each iteration

### The PHP do...while Loop

The do...while loop will always execute the block of code once, it will then check the condition, and repeat the loop while the specified condition is true.

### Syntax

do {  
*code to be executed;*} while (*condition is true*);

The example below first sets a variable $x to 1 ($x = 1). Then, the do while loop will write some output, and then increment the variable $x with 1. Then the condition is checked (is $x less than, or equal to 5?), and the loop will continue to run as long as $x is less than, or equal to 5:

<?php  
$x = 1;  
  
do {  
    echo "The number is: $x ;  
    $x++;  
} while ($x <= 5);  
?>

This example sets the $x variable to 6, then it runs the loop, **and then the condition is checked**:

<?php  
$x = 6;  
  
do {  
    echo "The number is: $x ;  
    $x++;  
} while ($x <= 5);  
?>

The PHP for Loop

The for loop is used when you know in advance how many times the script should run.

Syntax

for (*init counter; test counter; increment counter*) {  
  *code to be executed for each iteration;*  
}

Parameters:

* *init counter*: Initialize the loop counter value
* *test counter*: Evaluated for each loop iteration. If it evaluates to TRUE, the loop continues. If it evaluates to FALSE, the loop ends.
* *increment counter*: Increases the loop counter value

The example below displays the numbers from 0 to 10:

<?php  
for ($x = 0; $x <= 10; $x++) {  
    echo "The number is: $x ;  
}  
?>

Example Explained

* $x = 0; - Initialize the loop counter ($x), and set the start value to 0
* $x <= 10; - Continue the loop as long as $x is less than or equal to 10
* $x++ - Increase the loop counter value by 1 for each iteration

This example counts to 100 by tens:

<?php  
for ($x = 0; $x <= 100; $x+=10) {  
    echo "The number is: $x ;  
}  
?>

Example Explained

* $x = 0; - Initialize the loop counter ($x), and set the start value to 0
* $x <= 100; - Continue the loop as long as $x is less than or equal to 100
* $x+=10 - Increase the loop counter value by 10 for each iteration

### The PHP foreach Loop

The foreach loop works only on arrays, and is used to loop through each key/value pair in an array.

Syntax

foreach ($*array*as$*value*) {  
  *code to be executed;*  
}

For every loop iteration, the value of the current array element is assigned to $value and the array pointer is moved by one, until it reaches the last array element.

The following example will output the values of the given array ($colors):

<?php  
$colors = array("red", "green", "blue", "yellow");  
  
foreach ($colors as $value) {  
  echo "$value ;  
}  
?>

The following example will output both the keys and the values of the given array ($age):

<?php  
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");  
  
foreach($age as $x => $val) {  
  echo "$x = $val;  
}  
?>

### PHP Arrays

An array stores multiple values in one single variable:

<?php  
$cars = array("Volvo", "BMW", "Toyota");  
echo "I like " . $cars[0] . ", " . $cars[1] . " and " . $cars[2] . ".";  
?>

### What is an Array?

An array is a special variable, which can hold more than one value at a time.

If you have a list of items (a list of car names, for example), storing the cars in single variables could look like this:

$cars1 = "Volvo";  
$cars2 = "BMW";  
$cars3 = "Toyota";

However, what if you want to loop through the cars and find a specific one? And what if you had not 3 cars, but 300?

The solution is to create an array!

An array can hold many values under a single name, and you can access the values by referring to an index number.

### Create an Array in PHP

In PHP, the array() function is used to create an array:

array();

In PHP, there are three types of arrays:

* **Indexed arrays** - Arrays with a numeric index
* **Associative arrays** - Arrays with named keys
* **Multidimensional arrays** - Arrays containing one or more arrays

### Get The Length of an Array - The count() Function

The count() function is used to return the length (the number of elements) of an array:

<?php  
$cars = array("Volvo", "BMW", "Toyota");  
echo count($cars);  
?>

### Complete PHP Array Reference

For a complete reference of all array functions, go to our complete [PHP Array Reference](https://www.w3schools.com/php/php_ref_array.asp).

The reference contains a brief description, and examples of use, for each function!

### PHP Indexed Arrays

There are two ways to create indexed arrays:

The index can be assigned automatically (index always starts at 0), like this:

$cars = array("Volvo", "BMW", "Toyota");

or the index can be assigned manually:

$cars[0] = "Volvo";  
$cars[1] = "BMW";  
$cars[2] = "Toyota";

The following example creates an indexed array named $cars, assigns three elements to it, and then prints a text containing the array values:

<?php  
$cars = array("Volvo", "BMW", "Toyota");  
echo "I like " . $cars[0] . ", " . $cars[1] . " and " . $cars[2] . ".";  
?>

### Loop Through an Indexed Array

To loop through and print all the values of an indexed array, you could use a for loop, like this:

<?php  
$cars = array("Volvo", "BMW", "Toyota");  
$arrlength = count($cars);  
  
for($x = 0; $x < $arrlength; $x++) {  
    echo $cars[$x];  
    echo ";  
}  
?>

### PHP Associative Arrays

Associative arrays are arrays that use named keys that you assign to them.

There are two ways to create an associative array:

$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");

or:

$age['Peter'] = "35";  
$age['Ben'] = "37";  
$age['Joe'] = "43";

The named keys can then be used in a script:

<?php  
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");  
echo "Peter is " . $age['Peter'] . " years old.";  
?>

### Loop Through an Associative Array

To loop through and print all the values of an associative array, you could use a foreach loop, like this:

<?php  
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");  
  
foreach($age as $x => $x\_value) {  
    echo "Key=" . $x . ", Value=" . $x\_value;  
    echo ";  
}  
?>

### PHP - Multidimensional Arrays

A multidimensional array is an array containing one or more arrays.

PHP supports multidimensional arrays that are two, three, four, five, or more levels deep. However, arrays more than three levels deep are hard to manage for most people.

### PHP - Two-dimensional Arrays

A two-dimensional array is an array of arrays (a three-dimensional array is an array of arrays of arrays).

First, take a look at the following table:

|  |  |  |
| --- | --- | --- |
| Name | Stock | Sold |
| Volvo | 22 | 18 |
| BMW | 15 | 13 |
| Saab | 5 | 2 |
| Land Rover | 17 | 15 |

We can store the data from the table above in a two-dimensional array, like this:

$cars = array  
  (  
  array("Volvo",22,18),  
  array("BMW",15,13),  
  array("Saab",5,2),  
  array("Land Rover",17,15)  
  );

Now the two-dimensional $cars array contains four arrays, and it has two indices: row and column.

To get access to the elements of the $cars array we must point to the two indices (row and column):

<?php  
echo $cars[0][0].": In stock: ".$cars[0][1].", sold: ".$cars[0][2].".;  
echo $cars[1][0].": In stock: ".$cars[1][1].", sold: ".$cars[1][2].".;  
echo $cars[2][0].": In stock: ".$cars[2][1].", sold: ".$cars[2][2].".;  
echo $cars[3][0].": In stock: ".$cars[3][1].", sold: ".$cars[3][2].".;  
?>

We can also put a for loop inside another for loop to get the elements of the $cars array (we still have to point to the two indices):

<?php  
for ($row = 0; $row < 4; $row++) {  
  echo "<p><b>Row number $row</b></p>";  
  echo "<ul>";  
  for ($col = 0; $col < 3; $col++) {  
    echo "<li>".$cars[$row][$col]."</li>";  
  }  
  echo "</ul>";  
}  
?>

### PHP Connect to MySQL

PHP 5 and later can work with a MySQL database using:

* MySQLi extension (the "i" stands for improved)
* PDO (PHP Data Objects)

Earlier versions of PHP used the MySQL extension. However, this extension was deprecated in 2012.

### Should I Use MySQLi or PDO?

If you need a short answer, it would be "Whatever you like".

Both MySQLi and PDO have their advantages:

PDO will work on 12 different database systems, whereas MySQLi will only work with MySQL databases.

So, if you have to switch your project to use another database, PDO makes the process easy. You only have to change the connection string and a few queries. With MySQLi, you will need to rewrite the entire code - queries included.

Both are object-oriented, but MySQLi also offers a procedural API.

Both support Prepared Statements. Prepared Statements protect from SQL injection, and are very important for web application security.

### Open a Connection to MySQL

Before we can access data in the MySQL database, we need to be able to connect to the server:

<?php  
$servername = "localhost";  
$username = "username";  
$password = "password";  
  
// Create connection  
$conn = new mysqli($servername, $username, $password);  
  
// Check connection  
if ($conn->connect\_error) {  
    die("Connection failed: " . $conn->connect\_error);  
}  
echo "Connected successfully";  
?>

### Close the Connection

The connection will be closed automatically when the script ends. To close the connection before, use the following:

$conn->close();

### Insert Data Into MySQL Using MySQLi

After a database and a table have been created, we can start adding data in them.

Here are some syntax rules to follow:

* The SQL query must be quoted in PHP
* String values inside the SQL query must be quoted
* Numeric values must not be quoted
* The word NULL must not be quoted

The INSERT INTO statement is used to add new records to a MySQL table:

INSERT INTO table\_name (column1, column2, column3,...)  
VALUES (value1, value2, value3,...)

The following examples add a new record to the "MyGuests" table:

<?php  
$servername = "localhost";  
$username = "username";  
$password = "password";  
$dbname = "myDB";  
  
// Create connection  
$conn = new mysqli($servername, $username, $password, $dbname);  
// Check connection  
if ($conn->connect\_error) {  
    die("Connection failed: " . $conn->connect\_error);  
}  
  
$sql = "INSERT INTO MyGuests (firstname, lastname, email)  
VALUES ('John', 'Doe', 'john@example.com')";  
  
if ($conn->query($sql) === TRUE) {  
    echo "New record created successfully";  
} else {  
    echo "Error: " . $sql  . $conn->error;  
}  
  
$conn->close();  
?>

### Get ID of The Last Inserted Record

If we perform an INSERT or UPDATE on a table with an AUTO\_INCREMENT field, we can get the ID of the last inserted/updated record immediately.

In the table "MyGuests", the "id" column is an AUTO\_INCREMENT field:

We also echo the last inserted ID:

<?php  
$servername = "localhost";  
$username = "username";  
$password = "password";  
$dbname = "myDB";  
  
// Create connection  
$conn = new mysqli($servername, $username, $password, $dbname);  
// Check connection  
if ($conn->connect\_error) {  
    die("Connection failed: " . $conn->connect\_error);  
}  
  
$sql = "INSERT INTO MyGuests (firstname, lastname, email)  
VALUES ('John', 'Doe', 'john@example.com')";  
  
if ($conn->query($sql) === TRUE) {  
    $last\_id = $conn->insert\_id;  
    echo "New record created successfully. Last inserted ID is: " . $last\_id;  
} else {  
    echo "Error: " . $sql  . $conn->error;  
}  
  
$conn->close();  
?>

### Select Data From a MySQL Database

The SELECT statement is used to select data from one or more tables:

SELECT column\_name(s) FROM table\_name

or we can use the \* character to select ALL columns from a table:

SELECT \* FROM table\_name

### Select Data With MySQLi

The following example selects the id, firstname and lastname columns from the MyGuests table and displays it on the page:

<?php  
$servername = "localhost";  
$username = "username";  
$password = "password";  
$dbname = "myDB";  
  
// Create connection  
$conn = new mysqli($servername, $username, $password, $dbname);  
// Check connection  
if ($conn->connect\_error) {  
    die("Connection failed: " . $conn->connect\_error);  
}  
  
$sql = "SELECT id, firstname, lastname FROM MyGuests";  
$result = $conn->query($sql);  
  
if ($result->num\_rows > 0) {  
    // output data of each row  
    while($row = $result->fetch\_assoc()) {  
        echo "id: " . $row["id"]. " - Name: " . $row["firstname"]. " " . $row["lastname"];  
    }  
} else {  
    echo "0 results";  
}  
$conn->close();  
?>

Code lines to explain from the example above:

First, we set up an SQL query that selects the id, firstname and lastname columns from the MyGuests table. The next line of code runs the query and puts the resulting data into a variable called $result.

Then, the function num\_rows() checks if there are more than zero rows returned.

If there are more than zero rows returned, the function fetch\_assoc() puts all the results into an associative array that we can loop through. The while() loop loops through the result set and outputs the data from the id, firstname and lastname columns.

### Select and Filter Data From a MySQL Database

The WHERE clause is used to filter records.

The WHERE clause is used to extract only those records that fulfill a specified condition.

SELECT column\_name(s) FROM table\_name WHERE column\_name operator value

### Select and Filter Data With MySQLi

The following example selects the id, firstname and lastname columns from the MyGuests table where the lastname is "Doe", and displays it on the page:

<?php  
$servername = "localhost";  
$username = "username";  
$password = "password";  
$dbname = "myDB";  
  
// Create connection  
$conn = new mysqli($servername, $username, $password, $dbname);  
// Check connection  
if ($conn->connect\_error) {  
    die("Connection failed: " . $conn->connect\_error);  
}  
  
$sql = "SELECT id, firstname, lastname FROM MyGuests WHERE lastname='Doe'";  
$result = $conn->query($sql);  
  
if ($result->num\_rows > 0) {  
    // output data of each row  
    while($row = $result->fetch\_assoc()) {  
        echo "id: " . $row["id"]. " - Name: " . $row["firstname"]. " " . $row["lastname"];  
    }  
} else {  
    echo "0 results";  
}  
$conn->close();  
?>