

We are going to do the simple exercise of creating a database, then uploading the TriMet files from Activity 6. In the next activity, you will use SQL to answer the same questions you did with Excel about the TriMet data. Expect some frustration today in getting things to work. This is **normal**. There are known errors in the Excel file that will cause upload errors. One of the outcomes of this class is to help you overcome the learning curve and begin solving problems. Pay attention to syntax (i.e., upper/lower case letters, spelling, commas, quotes, spaces, etc.). Issuing commands to a computer by lines is always a little time consuming until you get the hang of a particular program. When something doesn't work, there is always a reason. It takes time, but it will be helpful if you learn to recognize error messages and work backwards to solve problems. Look for inconsistencies in something (e.g., Is that a word in my numeric field?). If I get really stuck, I always find that doing something incrementally helps. I start with something I know works first then start adding or modifying things until I find out what the problem is.

 <p><b>PURPOSE</b></p> <p>The purpose of this activity is to give you the opportunity to learn how to set up a database and upload data.</p>	 <p><b>LEARNING OBJECTIVE</b></p> <ul style="list-style-type: none"> <li>Transfer the Excel model of a simple database in Activity #2 to PostgreSQL. Learn to define tables, columns, rows, data types in the phpPgAdmin.</li> <li>Learn to navigate the phpPgAdmin tool.</li> </ul>
---	---

 <p><b>REQUIRED RESOURCES</b></p> <ul style="list-style-type: none"> <li>Web browser</li> <li>Files posted on class web site for Activity 6: <a href="#">stop_level.csv</a> &amp; <a href="#">stop_names.csv</a></li> <li>TriMet metadata description posted on the class website</li> <li>PostgreSQL data definitions: <a href="http://www.postgresql.org/docs/8.1/static/datatype.html">http://www.postgresql.org/docs/8.1/static/datatype.html</a></li> </ul>
--

 <p><b>TIME ALLOCATED</b></p>	<p>60 minutes in class</p>
--	----------------------------

## TASKS

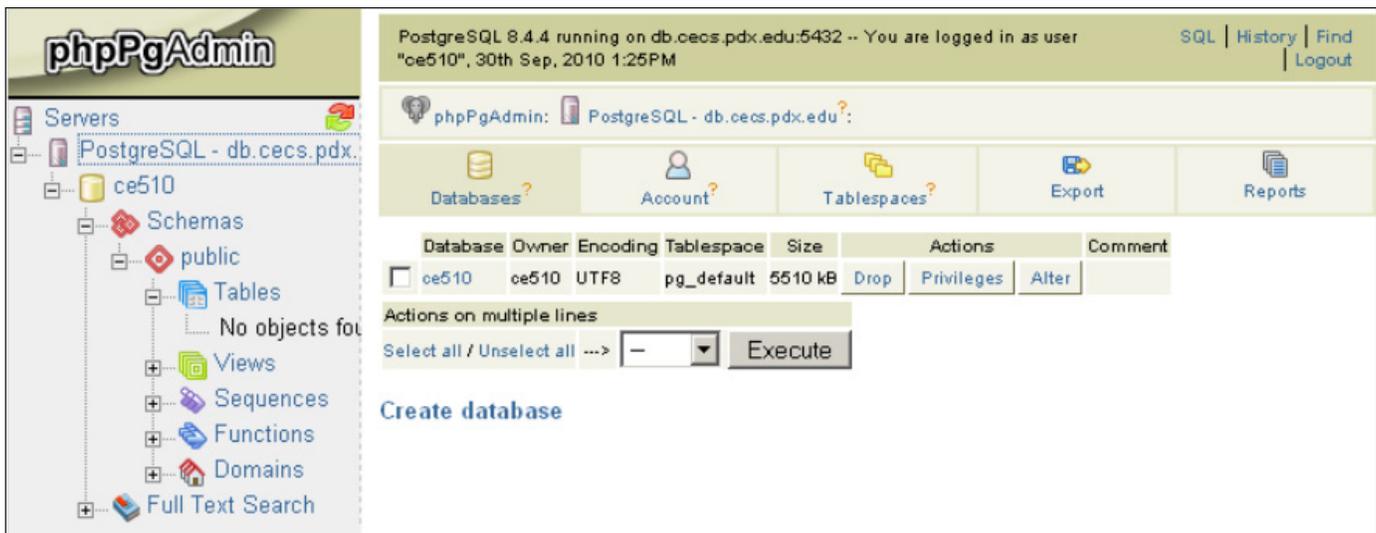


### A. Login to phpPgAdmin

Creating the database and table, managing data, uploading files, and other tasks to the database can be done many ways. We are going to start by using the web browser tool phpPgAdmin.

Login to your database: <https://cat.pdx.edu/phpPgAdmin/>

You should see something similar to Figure 7:

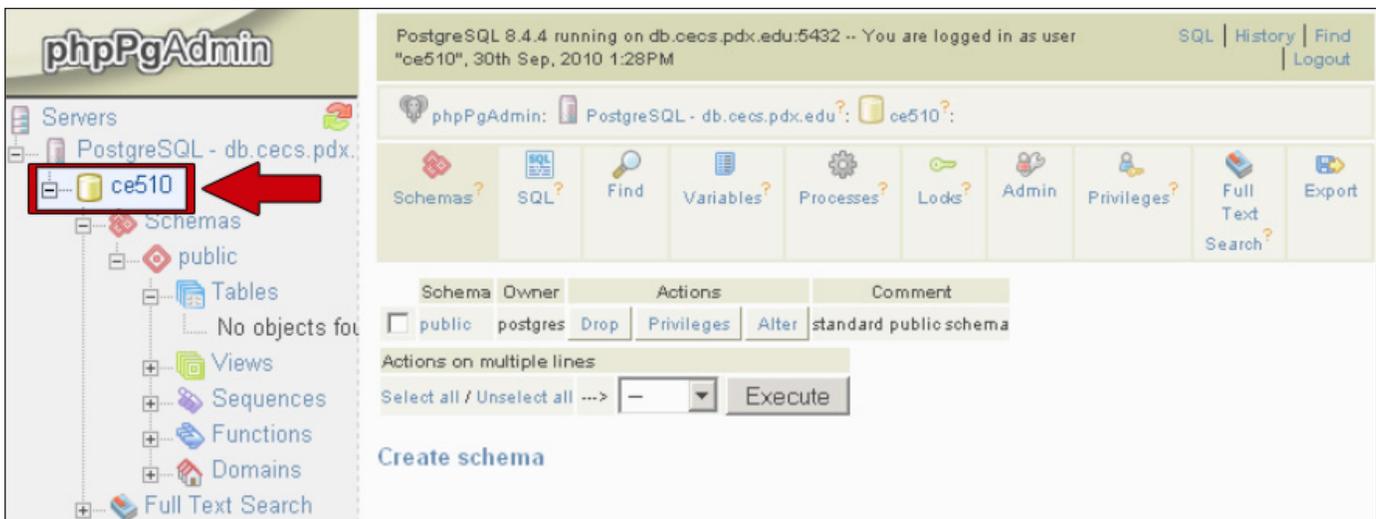


**Figure 7** Screen capture of the phpPgAdmin login screen.

This interface allows you to do some point/click navigating of the PostgreSQL database. You can also issue commands directly with the SQL interface. Spend a few minutes navigating this interface. When you are comfortable clicking around the software, let's get started.

## B. Select Your Database

Select your username listed under either “Database” in the mainframe, or under PostgreSQL – db.cecs.pdx.edu in the navigation window. You should see something similar to Figure 8 where the username is “ce510”:



**Figure 8** Screen capture of phpPgAdmin database screen

## C. Create A Schema

While not necessary or required, it is helpful to organize your database with a “filing” system. This filing system is called a *schema*.

Schemas are like creating folders in your operating system and help you to keep things better organized. In the PostgreSQL manual it says schemas are “are analogous to directories at the operating system level, except that schemas cannot be nested.”

PostgreSQL has a default schema called “public”. We could use that, but instead let’s create a schema *transit*.

First, click on



Then select Create Schema, type the name (*transit*), then **CREATE**. You should see on the left, under the public Schema that there is a new one called *transit*.

Click on the *transit* schema.

The syntax to refer to objects/tables/etc. that are filed under this schema is **transit.XXXX** where **XXXX** is the name of the object (e.g., table) we are referring to in SQL.

#### D. Type of Data

An important element in understanding data is to be able to define the types of data that is stored in each column. We will use the rather detailed PostgreSQL data definitions which can be found here: <http://www.postgresql.org/docs/8.1/static/datatype.html>.

Browse the table of contents, each major heading describes the general types. We will be most interested in:

- 8.1. Numeric Types
- 8.3. Character Types
- 8.5. Date/Time Types
- 8.6. Boolean Type

#### E. Create Table

Read each of these subheadings to get a feel for which data types can be defined. It can be rather tedious to go ahead and create all of these data types for all the fields by hand. In defining data types, as a general rule of thumb you should try to select the “smallest” data type that will work. For example, if an *int* (4 byte integer) definition will work, use it. Don’t select *bigint* (8 byte *int*) since in some data elements, the larger types take more disk space.

The primary components of the database are tables. You might want to think of tables as tabs in Excel spreadsheets. Data are organized in columns (that are named) and rows (called records). Let’s go ahead and use the interface to create a table for **stop\_names**. Be sure that you are in the schema *transit*.

Click on the **Create Table** (lower left of the screen) and follow the prompts.

Be careful not to specify a **LENGTH** for data types that don’t have this option. Refer to Table 8.1. Only those data types with [ ] in their description can have length or other options specified.

Browse the table. Notice that all column names are forced to lower case.

Notice the icons across the top (Columns, Indexes, Constraints, etc.—see Figure 9). We will come back to these later. Notice the buttons **BROWSE**, **ALTER**, **DROP**.



**BROWSE** lets you look at the data, **ALTER** lets you change properties about the column, and **DROP** removes it.

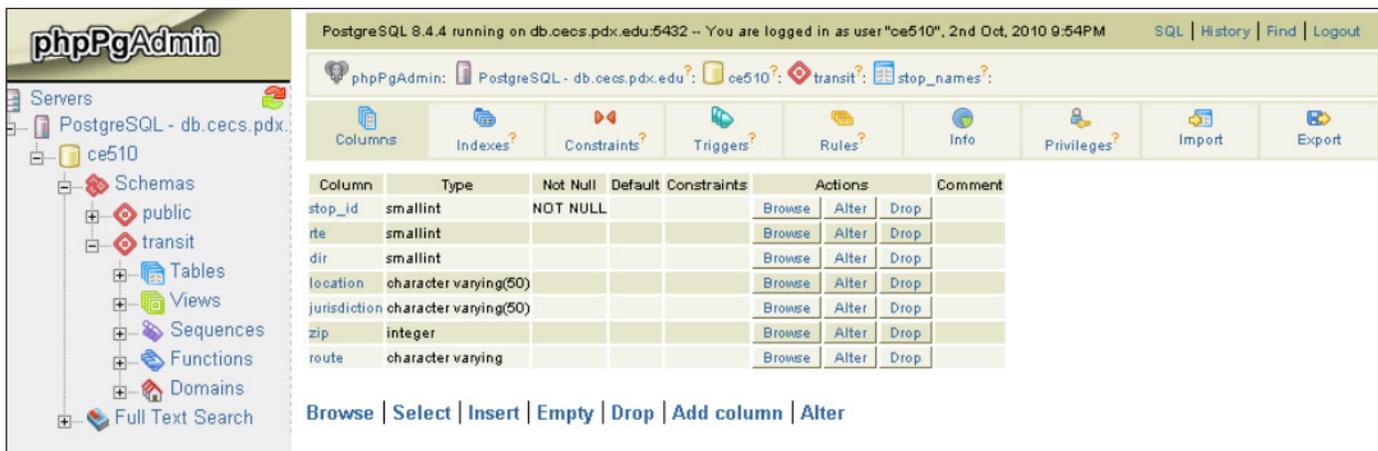


Figure 9 Screen Capture of the phpPgAdmin add table screen

## F. Import The Data

First, let's insert the first row of data from the CSV table manually. Click the **INSERT** option. Follow the prompts. Now use the **BROWSE** link to see the data you inserted.

Before we import data, we need to clear the data you just added to the database. Use the **EMPTY** link to purge all rows of the database (only the one row you created).

Now we are going to import the CSV file into the database. A good text editor is going to be very useful. My suggestion is to use Textpad. You can find it under "General Applications".

First, use the Textpad editor to verify the first row of columns names in the CSV file match **EXACTLY** the names of the columns your created. Change either the text file or **ALTER** the table to make the column names match.

It is good practice to use all lowercase for column names in SQL databases. So, while we are in the CSV, be sure to change the column names to lowercase. Textpad has a handy feature for making this conversion easy in the **Edit** menu. Save your edits. You can leave the file open in Textpad; the application doesn't lock the file from being read by other software, which is a nice feature!

Select the Import icon to upload the CSV file in the database:



You can leave the Format as **AUTO** and leave all other options as default. Navigate to the CSV file and click on **import**.

## G. Debugging

Error messages are going to be a fact of life when learning new computer programs, tools or writing algorithms yourself. If you received an error message, in most cases a careful inspection of the error message can give you a clue to what might have happened. The import runs a script that reads each line of the CSV then executes a **SQL INSERT** statement to put the data in the database. phpPgAdmin echoes the failed statement to the browser and tells you in blue which line failed to insert.

Therefore, if you get a statement back you know the program at least read the file. The error message will say something. In this case it gives you the row number where the data upload failed. Look at that row in the CSV that contains the error and ask yourself if something is different. What data mismatch error could there be?

The problem almost always is some data mismatch, either by the type you defined or a rogue data element in the file you are importing. This can be fixed by altering the column data types or by changing the data element.

Also, sometimes it is helpful to look at options for executing a function. Here, some of the upload problems can be solved by changing the import options from the default.

### H. Create and Import the Data for table **stop\_level**

Open the SQL code editor by clicking on the icon



Copy the SQL code into the code editor and choose execute, creating the table **stop\_level**. To import the **stop\_level** data, import the data the same way as the **stop\_names** table.

### DELIVERABLE

None; you will need the tables you inserted for the next activity.



### ASSESSMENT

Participation



