1. PostgreSQL + PLR

We are running the system under Windows environment. All the sql script is attached together with this report. Note: you should get R installed first.

1.1. **Installation.** The full description about the installation can be found at *http : //www.bostongis.com/PrinterFriendly.aspx?content_name = postgresql_plr_tut01*

1.2. **Importing the data as tables.** Once the installation is completed and the basic database is set(our name for the database is "test1"), we can start creating corresponding tables to import the source data:

1.Click the database "test1" inside the object browser and then click the "SQL" button on the main menu to open the sql window to type in queries.

2.Run the "create_table_allgrads.sql" in the window. When the query is done, an empty table is available in the object browser :"test1/schemas/public/Tables". Right click on the newly created table, and select import option.

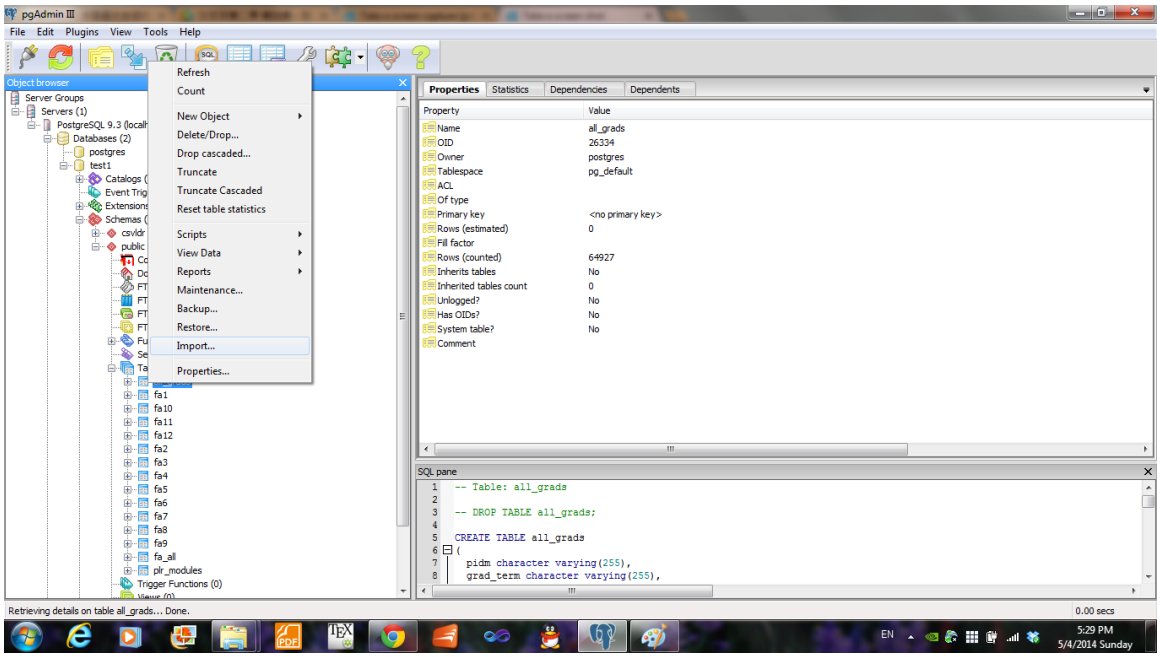

FIGURE 1. Import the data

3. Follow the instruction in the import window, you can now import the source data as a table easily.

4. Run the "create_table_fa.sql" in another sql window to create the table for the financial aid information and import the source data as stated above. Note: You may need to change the name of the table in the sql file each time if you want the 12 financial aid files as different tables.

5. You can also create the corresponding table by just right click the "Table" object inside the object window. There will be a "New table..." option.

1.2.1. *Problems met.* It did not go smoothly when we first imported the source data due to those bad entries in them. The good thing is the database will tell you which row and column in the source data that it can not import, so we directly goes to that entry and "correct" it.

1.3. **Writing PLR function.** Our function will be in the object "Function", all the functions can be combined with SQL clauses and used like the general sql functions.To create these function, just run the corresponding sql files in the sql window.

1.3.1. *Loading R modules.* Since we are using raw R functions as the body of our wrapper functions in the database. We require some R packages or libraries pre-loaded when we connect to the database. See PLR manual or our slides for details.

- Create the table to contain the library by running:
  CREATE TABLE plr_modules (
  modseq int4,
  modsrc text
  );
  (You just need to run it once, then the table will be seen in the object browser.)

- Insert into the table libraries you want:
  INSERT INTO plr_modules
  VALUES (0, 'library(Kendall)');

- Refresh the status by running: SELECT * from reload_plr_modules();

You now should see the "library Kendall" is in the table to use. To use the function in the library, you will have to write your own wrapper function.

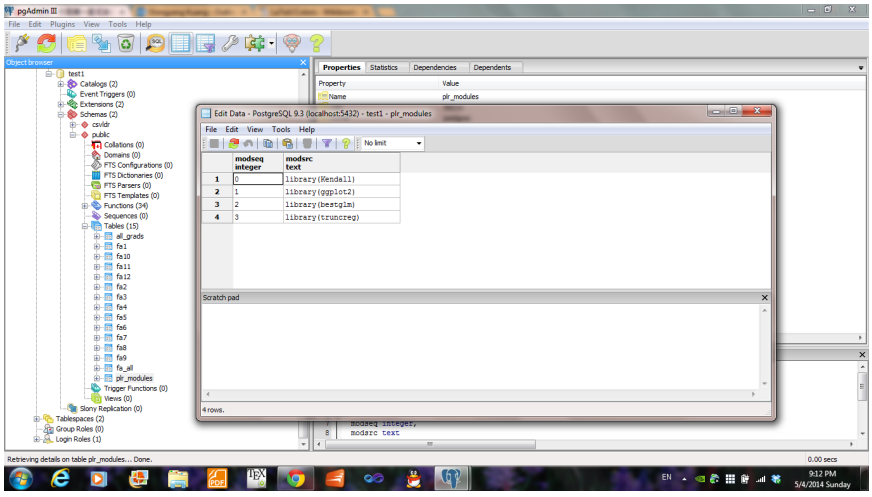

FIGURE 2. table of the R library

1.3.2. *Current functions.* Currently, we have made several R functions(prototypes) integrated into the data base:

- plr_plot(float[],float[]): Scatter plot of two columns;
- plr_qqplot(float[],float[]): Quantile-Quantile plot of two culumns;
- boxplot(float[]): Takes a numerical valued column generating a boxplot;
- hist(float []): Takes a numerical valued column generating the histogram of that column and showing frequencies of each subintervals;
- plr_qplot(float[], float[]): With proper parameters, it is a function that can make lots of fancy plots.By default, it is generates a histogram.
- median(float []): AGGREGATE function. It takes a numerical valued column showing the median;
- quantile(float [],float): AGGREGATE function. IT takes a numerical valued column and a float type number showing the corresponding quantile of the column;
- iqr(float[]): AGGREGATE function calculating the internal quantile range of certain variable;
- plr_corr_Kendall(float[],float[]): Takes two numerical valued columns calculating the Kendall's correlation between them.
- lm_plr(float[],float[]): Takes two numerical valued columns calculating the parameters of the linear regression (first argument v.s second argument).
- glm_plr(float[],float[]): Version of general linear regression;
- truncreg_plr(float[],float[]): Version of truncate linear regression;

1.3.3. *Before you use our functions.*

a) Before you run those linear regression function, make sure you have created the type "lm_type" in the database(the script is in lm_plr.sql), so please run lm_plr.sql first before you create other linear regression functions;

b) To use the non-aggregate function, please use "array_agg(column_name)" if the function's input is type "float[]"; E.G. SELECT median(uw_gpa) FROM all_grads, while you should use SELECT lm_plr(array_agg(uw_gpa),array_agg(first_term_gpa) FROM all_grads.

c) For a function that plots, it won't actually plot until you cross out the sql window. And the plot is by default saved in "D:/" in pdf format unless you change

it somewhere else by modifying the sql script for that function.

d) The functions we present here are actually simpler version of the functions in R by restricting the inputs and outputs. The intention was for users to use them without much knowledge in R. If user knows well enough about these functions in R, he or she can actually add parameters or see other output by altering parameters directly in the sql scripts creating those functions.

1.3.4. *Problems met.* The biggest problem we met in this part is the datatype-mapping between R and PostgreSQL. We have to understand the input and output structures in R, which varies depending on different functions. We also have to consider how to treat the output from the database so that R function can take them as input legally. For some the functions, in order to make it work, we restricted the R output and had to created corresponding composite type holding those outputs. For example, we created the lm_type and used it in our wrapper function to hold the coefficient part from the output of corresponding R function.

1.4. **Future Works.** Currently we just have very simple R functions integrated into the data base. On one side, we will try to migrate as many R functions as possible; on the other hand, we will try to make the integrated function more powerful by carefully study the data types and their mapping between our database and R.