**Dependency Manager**

Brennan Douglas

8/17/2017

Description:

This python program was developed when updating the ODOT TOCS Archive Process to stage tables in a specific order (Foreign Key relations copied first to their Primary Key counterparts last) that prevents concurrently streaming data from corrupting the archive database’s data integrity. This program calls this order the “copy order”, and the main algorithm for resolving this order from the internal dependency tree representation can be found in *copy\_order.py* files in the function *to\_copy\_order\_v2*.

Dependency manger provides a way to manually add tables to the dependency tree, automatically load them from a custom output file or the saved result from a provided SQL Query on a database with the results saved in a csv file. These can all be used to manage the loading and saving of the internal dependency representation. It also provides a mechanism to traverse the tree, though when the program starts it is in a void state outside of the dependency tree (the root state, accessed via the ***fl*** command) where any table can be entered into, from there only children and parents can be entered (via the ***i***[in – children] and ***u***[up – parents]).

General Usage:

* Managing current state:
  + On starting the program the *.saved\_session* file is loaded into the internal structure, so if it is not empty the program will not start in a clean state.
  + Exiting the program using the standard **quit** command, ***q***, saves the current *.saved\_session* file to the *.saved\_session\_old* file, dumps the internal structure then overwrites the *.saved\_session* file with a blank file.
  + Exiting the program using the **save and quit** command, ***sq***, overwrites the *.saved\_session* file with the current internal structure.
  + The internal structure can be saved at any time overwriting the *.saved\_session* file using the **save session** command, ***ss***.
* Starting from an empty state:
  + If the program has never been used before (*.saved\_session* is empty) then a dependency structure needs to be loaded into it, this is accomplished using the **load** command, ***l***. A file is passed to this command, by default it accepts a file that is formatted as the **save** command, ***s***, outputs but by adding the *--sql* flag before the filename it will accept a csv file that is the saved results on the database being modeled from the provided SQL Query.

Steps on how to Generate the Copy Order:

1. Run the provided SQL Query (*SQLQuery\_PullTableDependencies*) on the database that the order will be generated on and save the results to a csv file.
2. Run the *main.py* file, then dump the current internal structure with “***ds***” so that it is a clean slate. Next, load the csv file in with the command “***l --sql <csv results file>***”. To check to make sure it is loaded run “***fl***” to go to root, then run “***lp***” or “***lc***” to see all the imported tables. Finally, save the current structure to the *.saved\_session* file with “***ss***”.
3. Now the circular dependencies need to be removed. First, check if any circular dependencies exist with “***circle***”. If they do there are two options, manually traverse the table structure and delete the desired links yourself, or run “***circle -df***” or “***circle -db***” which with either delete the front or back link of each of the circular references in its internal memory. Finally, save the current structure to the *.saved\_session* file with “***ss***”.
4. Finally, the copy order can be determined. Run “***co***” to have the copy order printed out to the terminal or run “***sco***” to generate the copy order to a specified file.