CSI3130/3530 – Setting up PostgreSQL

1 Introduction to PostgreSQL

PostgreSQL is an object-relational database management system based on POSTGRES, which was developed at the University of California at Berkeley. PostgreSQL is an open-source descendant of this original Berkeley code.

2 Installation of PostgreSQL on Your Own Machine

This section describes how to download, install and use PostgreSQL version 8.1.4 in a Linux environment. This is the version that will be used for the CSI3130/3530 project. Newer versions were changed internally, especially from versions 9.xx on. I never had time to update this material to those newer versions: it is future work. A compressed tar archive containing the PostgreSQL version 8.1.4 source code can be downloaded via the Web though the link that you learned from the TA in the labs. For a more complete installation guide, refer to the INSTALL file, which is included with the PostgreSQL source code. Youhave already learned from your TA how to install PostgreSQL on your own machine,. Here I am just reminding you of how to do so. .

Once you have obtained a copy of postgresql-8.1.4.tar.gz, you can unpack it like this:

tar xzf postgresql-8.1.4.tar.gz

This will create a new directory called postgresql-8.1.4 containing a complete copy of the PostgreSQL source code.

PostgreSQL is large. The compressed tar file is approximately 19Mb. You will need about 110Mb of space to hold the uncompressed source code once it has been unpacked by the above command. After unpacking the source code, you can delete the tar.gz file.

Next, cd into the postgresql-8.1.4 directory and configure PostgreSQL as follows:

./configure --enable-debug --enable-cassert

The two configuration arguments, --enable-debug and --enable-cassert, are used to enable debugging of PostgreSQL code and assertions, respectively. While configuring PostgreSQL, you may find that some dependencies are missing. In particular, you may need to install the readline library. The following warning comes from the PostgreSQL INSTALL file:

“If you are using a package-based Linux distribution, be aware that you need both the readline and readline-devel packages, if those are separate in your distribution. ”

If you are not using a package-based distribution, you can download and build the library directly from

http://ftp.gnu.org/gnu/readline/readline-5.0.tar.gz

Once PostgreSQL is successfully configured, build it by running

make

Be sure that the make that you are using is GNU make, since this is what PostgreSQL expects. On some systems, this is called gmake, rather than make. If you are not sure what kind of make you have, try running make --version, and check whether it reports that it is GNU make.

Next, install PostgreSQL with the command:

make install

Note that you will probably need to run this installation command as root. Root privilege is required inorder to install PostgreSQL into the default location, which is /usr/local/pgsql. You can become root using the su command before running make install. This is the only step that requires root privileges, so you can exit your root shell after this step.

At this point, PostgreSQL has been built and installed. Before running any of the PostgreSQL programs below, you will need to add /usr/local/pgsql/bin to your shell's command search path, so that it can find the newly installed programs.

The next step is to create a directory to host the database:

initdb -D $HOME/pgdb

This will initialize the database in a pgdb directory under your home directory. If you wish, you can choose a different directory name.

You should now be able to start the database server by executing the command

postmaster -D $HOME/pgdb

It is a good idea to launch postmaster in a separate window. To kill the server, you can simply type control-C in the postmaster window. Otherwise, you will need to use a command like this:

kill -INT 'head -1 $HOME/pgdb/postmaster.pid'

to kill the server. This works because the file $HOME/pgdb/postmaster.pid contains the process identifier of the postmaster process. Note that the quotes in the above command are backquotes – this is important.

To create a new database named dbname, use the command

createdb <dbname>

To execute SQL commands, you have to run psql, an interactive PostgreSQL client program:

psql <dbname>

You can use the psql client to interactively create tables, insert data, and issue queries. A sample script that creates two tables and performs a number of queries can be found at

postgresql-8.1.4/src/tutorial/basics.source
To quit the interactive client, use the psql command \q.

3 Modifying PostgreSQL Source Code

Course assignments will require that you add or modify PostgreSQL source files. Before modifying PostgreSQL files, make sure that you have a backup copy of the original file so that you can always undo your modifications. Note that after making changes to PostgreSQL files, you should clean the built version using

make clean

before rebuilding from the modified source code. This is particularly important if you have modified header files. Before each assignment, you should start with a fresh copy of the PostgreSQL source code.

4 Debugging PostgreSQL

PostgreSQL is a client/server system, meaning that a user runs a client process, like the psql commandinterpreter, which talks to a postgres server process. The main PostgreSQL server, called postmaster, spawns a separate postgres server process for each client connection.

There are two main methods that can be used for debugging. The first method is to print out debugging information (e.g. variables' values) from within the server process. The second method is to use a debugging facility to insert breakpoints at interesting locations and inspect the variables' values and the flow of control.

4.1 Printing Server Debugging Information

If you put printf()'s in your server code, they will not be visible at the psql client side. PostgreSQL provides a special function called elog() to get messages from the postgres server process to appear at the client. To insert debugging statements in your code, you should use the elog() function, with the first argument being DEBUG1. Note that elog() takes a message string as its main argument; to construct such a string you may want to use the sprintf() routine.

4.2 Using a Debugger

You may use any available debugger, such as gdb, to debug PostgreSQL server code. To start debugging a PostgreSQL server process on a local machine, you first need to startup the server (i.e., postmaster), and the client (i.e., psql). Then, you must attach the debugger to the PostgreSQL server process that is serving your psql client. To do this using gdb, open another shell window on the same host on which your PostgreSQL server is running, and enter the command:

ps -af | grep <userid> | grep postmaster
where <userid> is your user id. This will give you a list of your PostgreSQL server processes, which should look something like this:

<userid> 19007 18984 0 00:02:38 pts/38 0:00 postmaster -D <db>
<userid> 18987 18984 0 00:02:22 pts/38 0:00 postmaster -D <db>
<userid> 18984 17200 0 00:02:21 pts/38 0:00 postmaster -D <db>
<userid> 18989 18988 1 00:02:22 pts/38 0:00 postmaster -D <db>
<userid> 18988 18984 0 00:02:22 pts/38 0:00 postmaster -D <db>

Each line corresponds to a process. The second entry on each line is a process id, and the fifth entry on each line is the time at which the process started. You want to identify the process id of the postmaster process with the latest start time. This is the PostgreSQL server that is serving the psql client that you just started. In the above example, the desired server process id is 19007, which started running at time 00:02:38.

Once you have identified the correct process id, launch the debugger:

gdb postgres

At the gdb command prompt, enter

attach <process-id>

where <process-id> is the PostgreSQL server process id that you just identified: 19007 in the example above. Attaching gdb to the PostgreSQL server process will cause the server process to pause, so that you can use the debugger to inspect code and variables, set breakpoints, and so

on. Issue gdb's continue command when you are ready to let the server process continue running. If you wish to exit gdb without killing the PostgreSQL server process, you can issue a detach command to gdb.

5 Documentation

The main source for PostgreSQL information is the official documentation at

http://www.postgresql.org/docs/8.1/static/index.html.

In the source code, you will find README files within each component directory (e.g. parser, executor and optimizer components). Comments found in the PostgreSQL code are particularly helpful in understanding how PostgreSQL functions are implemented.