ChatScript PostgreSQL

© Bruce Wilcox, gowilcox@gmail.com brilligunderstanding.com

Revision 4/30/2016 cs6.54

ChatScript ships with code and WINDOWS libraries for accessing PostgreSQL but you need a database somewhere. All builds with the postgres client are `ChatScriptpg` in naming as opposed to `ChatScript.exe` or `LinuxChatScript32` or `LinuxChatScript64` which are non postgres builds.

On Linux, ChatScript ships with PostgreSQL client built-in, but if you want to rebuild the system for some reason, then while standing in /src type either

`make server` – build ordinary CS server w/o PostgreSQL client `make pgserver` – build CS server w PostgreSQL client (you had to install postgress first)

Usually postgres somes with amazon servers, but if not you may have to do something like

```
sudo apt-get install libpq-dev
```

to build successfully.

On Mac and IOS `#define DISCARDDATABASE` is on by default. If you have access to a PostgreSQL server remotely, that's fine. If you need one installed on your machine, see the end for how to install one.

Using Postgres as file server

Aside from using Postgres to store data for a chatbot to look up, one can also use Postgres as a replacement for the local file system of the USERS directory. This allows you to scale CS horizontally by having multiple CS servers all connecting to a common Postgres DB machine so a user can be routed to any server. To do this, on the command line use:

```
pguser="hostaddr = 129.22.22.24 port = 5432 user = postgres password = somepassword "
```

You can replace hostaddr with host to use a named notation. If both are omitted, it defaults to localhost. Localhost might not work as the name of the host, in which case use 127.0.0.1 .

You do not specify the dbname. The system assumes you have a db named `users`. If it doesn't find it, it will try to open a root database named `postgres`. If it can find that, it will then create the `users` db.

Obviously put the correct data for your postgres machine. CS will automatically create a `users` database and `userfiles` and `userlogs` and `userbugs` tables. Each user will get an entry in the userfiles table and log entries will be stored in the userlogs table instead of going into the server logs. Similarly any bugs

detected will be posted to the central database instead of the LOGS folder locally. If Postgres is not available, the CS server will quit. Hopefully you have an automatic restart cron job and it will be able to connect to Postgres the next time.

Access A PostgreSQL Database

There are only a couple of functions you need.

^dbinit(xxx)

xxx is the parameter string you want to pass PostgreSQL. The default minimal would be something like this:

```
u: (open)
    if (^dbinit( dbname = postgres port = 5432 user = postgres password = mypassword ))
        {db opened}
    else
        {db failed}
```

Once the named database is open, it becomes the current database the server uses until you close it no matter how many volleys later that is. Currently you can only have one database open at a time. By the way, if a call to ^dbinit fails, the system will both write why to the user log and set it onto the value of $$db_error.

^dbinit(^"null")

creates a dummy database access, whereby nothing is actually done thereafter but you can pretend to write to the database.

^dbinit will fail if the database is already open. If you want it return without failure, having done nothing, just add this at the start of your arguments: EXISTS

```
u: (open)
    if (^dbinit(EXISTS dbname = postgres port = 5432 user = postgres password = mypassword )
```

^dbexecute(“xxx” 'myfunc)

executes database commands as a blocking call.You may omit the 2nd argument if those commands do not generate answers, otherwise it names a function you have defined whose argument count matches the number of pieces of information coming from the database.

Typically a database query like SELECT will return multiple rows containing multiple columns of values. The number of values per row must match your

function, which is called once for each row. The exception is that a one-argument function will accept all the values returned from the query for each row, automatically concatenated with underscore separators.

If any call to your function fails, `^dbexecute` stops and fails. When you want to use multiple commands to the database, you must separate them with ; and only answers from the last command are returned (per PostgreSQL standards and behavior).

Note: for any direct calls, the format string `^"xxx"" is your friend, as it will generally preserve the contents of the string unchanged (except for variable substitutions) to pass to`^dbexecute`. In addition, don't worry generally about single quotes inside of quoted parameters. ChatScript will automatically double the internal single quotes for you. There will be issues with format strings because ChatScript finds meanings in some things that conflict with postgres command structure. "‘ if (^dbexecute(^"INSERT INTO word VALUES '_1');" NULL)) {word added} else {dbexecute failed - $$db_error} "‘ The_1has issues because postgre wants quotes around string values, but ChatScript will be trying to decode'_1as the original form of_1. This might require you to do this: "‘ $$tmp = '_1 if (^dbexecute(^"INSERT INTO word VALUES '$$tmp');" NULL)) {word added} else {dbexecute failed - $$db_error} "‘ Due to conflicts in interpreting‘,*youcan'tusepostgres‘value*`$notation inside a quoted string because ChatScript will seek variable values. So, after this, you can read whatever commands PostgreSQL supports because they all funnel through`^dbexecute‘.

Note there is a similar issue with text strings coming back from Postgres. They may be multiple words, etc. You should declare arguments to outputmacros that will receive postgres text like this:

```
outputmacro: ^myfunc( ^arg1.HANDLE_QUOTES ^arg2)
```

where `^arg1` will receive a text argument. This causes DBExecute to put double-quotes around the value (and \ on any interior quotes) and then the call to `^myfunc` will strip them off after evaluation. You can use `:trace sql` to see what goes in and what comes out. By the way, if a call to `^dbexecute` fails, the system will both write why to the user log and set it onto the value of `$$db_error`.

PostgreSQL Commands

The basic commands you most likely need are:

Creating Data

```
CREATE DATABASE name - to make a new database
CREATE TABLE weather - to define a new table
(
city varchar(80),
stateAbbr char[2],
temp_lo int,
temp_hi int,
prcp real,
date date
)
```

```
DROP TABLE weather
```

to delete a table

Note that the weather table, while shown all nicely laid out, has to be a single string to `^dbexecute` which will NOT span multiple lines (but could have been created by `^join(...)` whose arguments could have been neatly laid out. Field names are case insensitive (per std).

```
INSERT INTO weather VALUES ('San Francisco', 'CA', 46, 50, 0.25, '1994-11-27')
```

Note that text values must be encased in 'xxx' (per std). Better style is to name the fields and values, omitting any you don't want to supply:

```
INSERT INTO weather (date, city, temp_hi, temp_lo)
VALUES ('1994-11-29', 'Hayward', 54, 37)
```

or with lots of data and a file on the same machine as the database server,

```
COPY weather FROM '/home/user/weather.txt'
```

Retrieving Data

```
SELECT * FROM weather
```

retrieves all fields and all rows of table weather (rows in arbitrary order). You can name the fields and order with this:

```
SELECT city, temp_lo, temp_hi, prcp, date FROM weather
```

assuming the function of yours named expects 5 arguments. And a constrained query like:

```
SELECT * FROM weather WHERE city = 'San Francisco' AND prcp > 0.0
```

And of course there are other things the database can do, including joins and other searches, but you can go read the PostgreSQL manual for that. The above is enough to do many simple things.

Creating A PostgreSQL Database

It's never as easy as you'd hope. Maybe I've hit all the stumbles for you.

Installation on Windows

While in theory you can install a 64-bit version of PostgreSQL on a 64-bit machine, I found there were problems in compiling ChatScript with it, so I use only the 32-bit version, which works on either 32-bit or 64-bit Windows machines. Note that the 32-bit dll that is needed is at the top level of ChatScript, so the system will automatically find it. For reference, the compilation header files and library for loading are in `src/postgre/WIN32`

First, you need to download the 32-bit Windows PostgreSQL. I used version 9.3.3 from this page: http://www.enterprisedb.com/products-services-training/pgdownload#windows win-x86-32. Download that and run it. You can default everything. ChatScript will just work with it.

Installation on Mac

Sorry. Don't have a mac. You have to figure it out yourself.

Installation on Linux

So many choices. You probably have to read docs for your system. I went wrong in so many directions I lost track of whether I was logged in as me, su, sudo, postgres, etc. If you have problems installing, here is my best knowledge... After that go search the internet or documentation, don't come to me to get postgres installed.

On my amazon AMI server I used

```
sudo yum -y install postgresql postgresql-server postgresql-devel postgresql-contrib
```

My 64-bit machine worked fine, my 32-bit machine had issues. Nominally you need to: set a password for your postgres user (the user that runs the db) and initialize the database in PGDATA via

```
service postgresql initdb
```

or

```
su - postgresql
initdb -D /var/lib/pgsql9/data/
```

Confirm there is now stuff installed in `/var/lib/pgsql9`. You may need to edit `/var/lib/pgsql9/pg_hba.conf` to change the local validation from peer to trust - e.g.

```
local all all trust
```

Sometimes it was already trust, and sometimes it was peer. On a 32-bit AMI I tried to lauch the postgres server but it wanted it to have been installed in `/var/lib/pgsql` for some reason. So I had to create a symbolic link to make it happy:

```
ln -s /var/lib/pgsql9 /var/lib/pgsql
#2 launch the server via
service postgresql start
```

or

```
su - postgres - log in as postgres user
pg_ctl start - starts server in background
#3 make it auto restart on reboot
chkconfig postgresql on
```

You can confirm a server is running by typing

```
psql
```

which is a tool to log into the server. You may have to be logged in as postgres. There's a bunch more about setting up login passwords, etc. RTFM.