

Lab 2: PostgreSQL Tutorial II: Command Line

In the lab 1, we learned how to use *PostgreSQL* through the graphic interface, *pgAdmin*. However, *PostgreSQL* may not be used through a graphical interface. This tutorial will tell you how to manipulate the *PostgreSQL* database through shell command line.

Objectives

The goals for you to take away from this lab are:

- Get familiar with *PostgreSQL* shell commands;
- Learn the procedure of using shell commands and SQL statements to create, manipulate, backup and restore databases;
- Practice methods for loading data, querying and deleting data in a database.

Data and SQL scripts

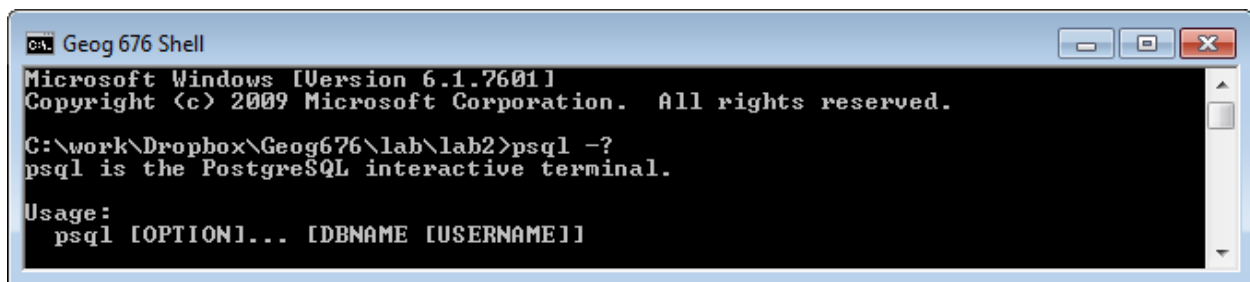
Please download the data and scripts (Lab2.zip) in the Lab section at Learn@UW system. If you are using Windows, put it under C:/lab2/ (if you do not have a C:/lab2 folder, please create one). Unzip the data.

Please go through Part I: Exercise Tutorial and then finish tasks listed in Part II: Lab Assignment.

Part I: Exercise Tutorial

1. Getting a PostgreSQL command prompt

- ➔ To get a command shell that is set up to run these tools on the Lab machines, go to the file directory (e.g., c:/lab2/, in this case), run the PostgreSQL shell launcher script with file name "start_postgresql_shell.bat". You should only run this script from a windows machine (or some computer set up the same way) by double clicking the script.
- ➔ To test that the script has set up the tools paths correctly, type **psql -?** at the prompt, and hit Enter. You should see a usage message similar to the following:



```
Geog 676 Shell
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\work\Dropbox\Geog676\lab\lab2>psql -?
psql is the PostgreSQL interactive terminal.

Usage:
psql [OPTION]... [DBNAME [USERNAME]]
```

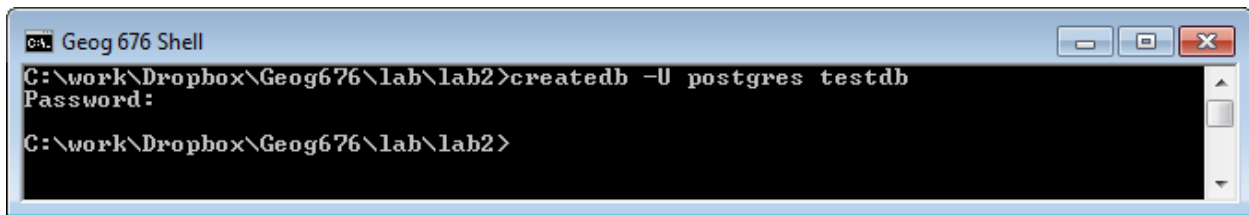
2. Creating/Deleting a PostgreSQL database

Just as in other DBMS, such as SQL Server, your tables must be placed in a database, which you must create. To create a database named *testdb*, do the following:

- 1) Make sure PostgreSQL is running (by default, PostgreSQL is running once Windows boots up), and open a PostgreSQL shell as above.

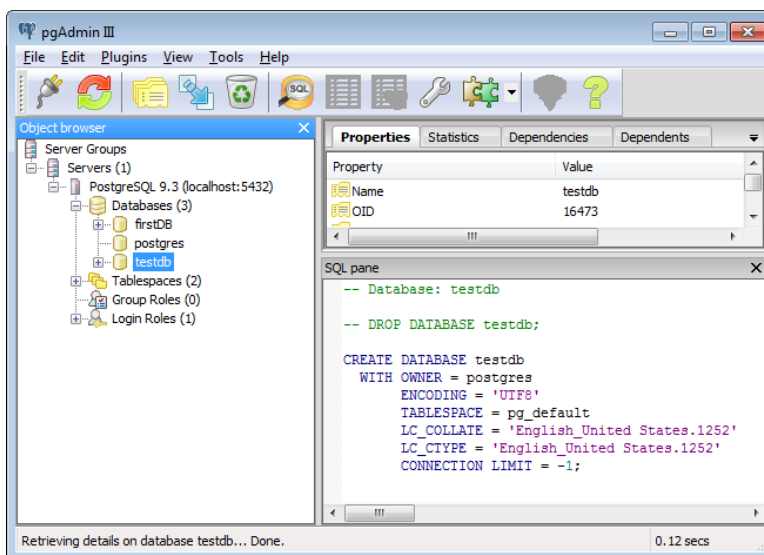
2) Use the `createdb` command to create the database (enter “postgres” when prompt for password):

```
>createdb -U postgres testdb
```



```
C:\work\Dropbox\Geog676\lab\lab2>createdb -U postgres testdb
Password:
C:\work\Dropbox\Geog676\lab\lab2>
```

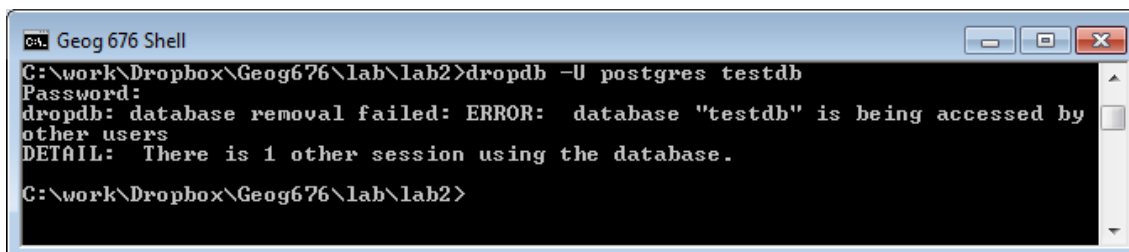
3) Now, if you open pgAdmin III and connect to the local database server “PostgreSQL 9.3 (localhost:5432)”, you will find that the testdb was created.



4) To delete (or drop) a database named `testdb`, do the following:

```
> dropdb -U postgres testdb
```

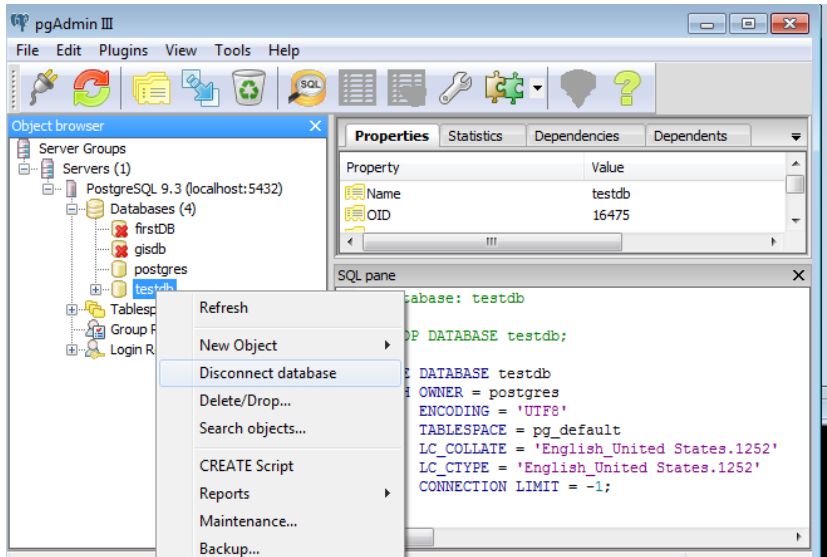
However, you may get the following error.



```
C:\work\Dropbox\Geog676\lab\lab2>dropdb -U postgres testdb
Password:
dropdb: database removal failed: ERROR:  database "testdb" is being accessed by
other users
DETAIL:  There is 1 other session using the database.
C:\work\Dropbox\Geog676\lab\lab2>
```

This means that your `pgAdmin III` is still connecting and using this database. So you should disconnect it first by:

➔ Right click the database and select **Disconnect database**



Then you can try to delete *testdb* again.

PostgreSQL is case-sensitive when looking up database and table names. However, PostgreSQL automatically lowercases all names given in SQL code, so for you, the case-sensitivity should only affect non-SQL code that needs to know the database name, including:

- The *createdb* command;
- The *psql* SQL query tool (discussed below);
- Any Java or other program code that connects to the database.

In such code, always spell the database name exactly as you created it, respecting case.

Question 1 (5 pts): Create a database named “gisdb” with *createdb* command. Take a screenshot of your command line window, and a screenshot of pgAdmin III showing the created database.
Tip: Alt + PrtScn can get you a screenshot of the current window.

3. Starting the SQL Command Line Terminal (psql)

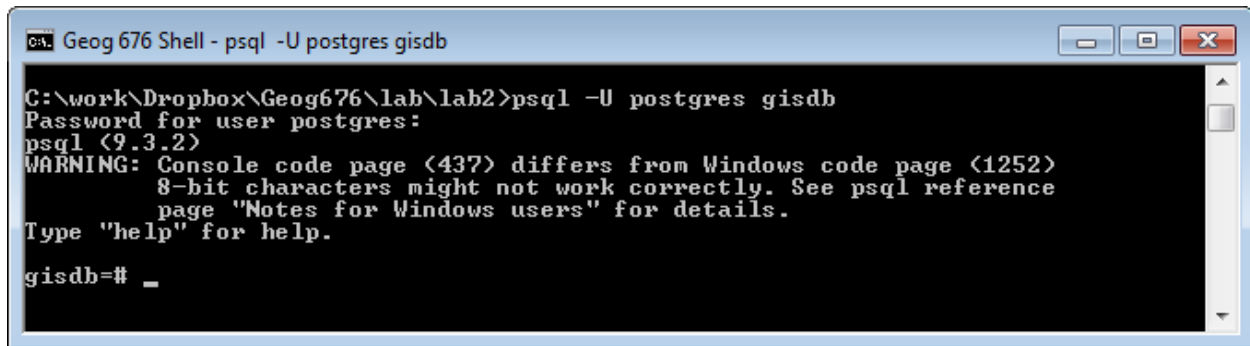
psql is a terminal-based front-end to PostgreSQL. It enables you to type in queries interactively, issue them to PostgreSQL, and see the query results. Alternatively, input can be from a file. In addition, it provides a number of meta-commands and various shell-like features to facilitate writing scripts and automating a wide variety of tasks. The following steps describe how to start the **psql**.

➔ Once PostgreSQL is running, you can run **psql** by opening a PostgreSQL shell command prompt through the method mentioned in **Section 1 “Getting a PostgreSQL command prompt”** and typing:

```
>psql -U postgres gisdb
```

where **postgres** is the user account (in this case the database superuser, you can always change to other user account). Being a superuser means that you are not subject to access controls. For the purposes of this tutorial that is not of importance. **gisdb**, created in the previous section, is the name of the database

you want to use. Of course, you can also use “firstDB” as the database name (If you omit the database name, psql defaults to accessing the database “postgres”).

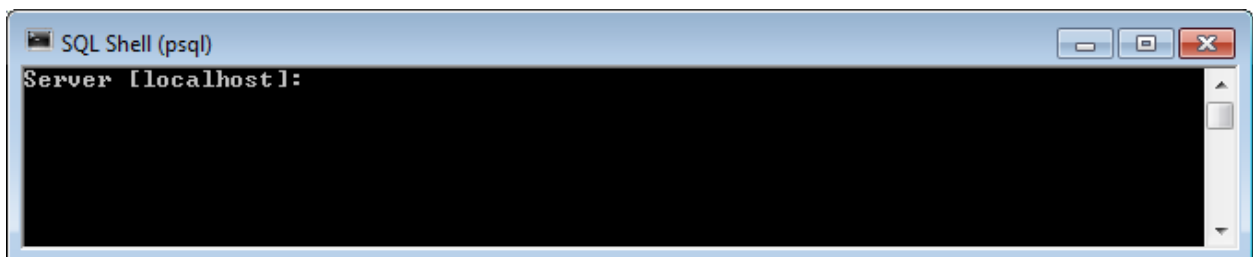


```
Geog 676 Shell - psql -U postgres gisdb
C:\work\Dropbox\Geog676\lab\lab2>psql -U postgres gisdb
Password for user postgres:
psql (9.3.2)
WARNING: Console code page (437) differs from Windows code page (1252)
         8-bit characters might not work correctly. See psql reference
         page "Notes for Windows users" for details.
Type "help" for help.
gisdb=# _
```

The line `gisdb=#` is the prompt for SQL statements which are sent to the database server, or non-SQL commands interpreted by `psql`. It indicates that `psql` is listening to you and that you can type SQL queries into a work space maintained by `psql`. Here, "gisdb" again is the name of the database; it may differ on your system if you use different database name.

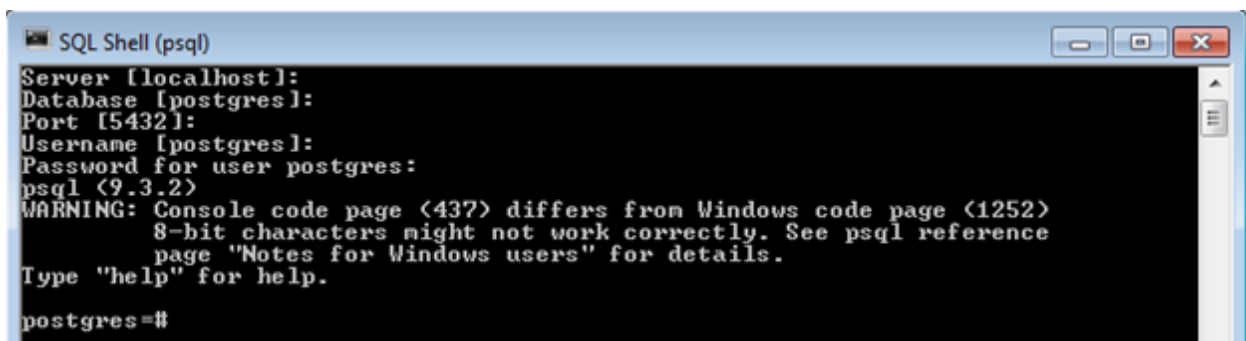
In addition, you can also start the `psql` through:

- Click Windows **Start**, select **Program-> Development-> PostgreSQL 9.3 -> SQL shell (psql)**. The SQL command line terminal opens.



```
SQL Shell (psql)
Server [localhost]:
```

- Enter the information for the successive prompts or accept the defaults as below by hitting **Enter** key.



```
SQL Shell (psql)
Server [localhost]:
Database [postgres]:
Port [5432]:
Username [postgres]:
Password for user postgres:
psql (9.3.2)
WARNING: Console code page (437) differs from Windows code page (1252)
         8-bit characters might not work correctly. See psql reference
         page "Notes for Windows users" for details.
Type "help" for help.
postgres=#
```

- Try out these commands in PostgreSQL shell command prompt:

```

CA: Geog 676 Shell - psql -U postgres gisdb
gisdb=# select version();
          version
-----
PostgreSQL 9.3.2, compiled by Visual C++ build 1600, 64-bit
(1 row)

gisdb=#

```

The psql program has a number of internal commands that are not SQL commands. They begin with the backslash character, "\". Here are some common commands you can practice first before you go to the next section.

Commands	Description	Commands	Description
\?	Get a help message	\do	List operators
\h <cmd>	help on a SQL command; replace<cmd> with the actual command	\i <filename>	Execute commands read from the file name <filename>
\q	Quit psql	\c <dbname>	connect to the db with name as <dbname>
\l	list all database	\r	Reset the buffer(discard any typing)
\dt	list all the tables	\dT	List types

As the message suggests, you can get help on the syntax of various PostgreSQL SQL commands by typing \h. You can exit psql by typing \q and hitting Enter (note the lack of a semicolon; this is required because \q is not an SQL statement and is not interpreted at the server).

```

CA: Geog 676 Shell
gisdb=# \q
C:\work\Dropbox\Geog676\lab\lab2>

```

\l can list all database as below.

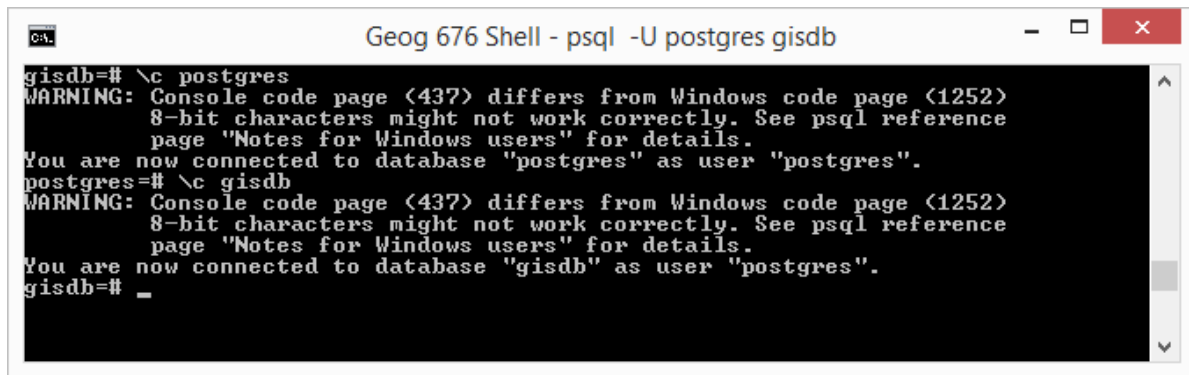
```

CA: Geog 676 Shell - psql -U postgres gisdb
gisdb=# \l
          List of databases
  Name      | Owner   | Encoding | Collate      | Ctype
-----+-----+-----+-----+-----
 firstDB   | postgres | UTF8     | English_United States.1252 | English_United S
 tates.1252 | postgres | UTF8     | English_United States.1252 | English_United S
 -- More --

```

Here we can see that “gisdb” we created before is included as well.

\c can connect to different database.



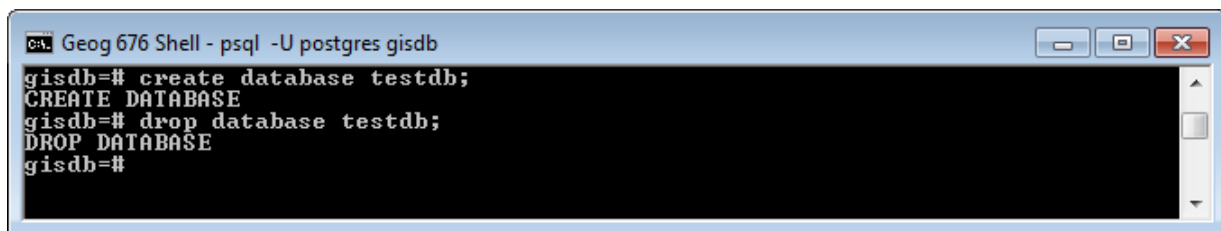
```
Geog 676 Shell - psql -U postgres gisdb
gisdb=# \c postgres
WARNING: Console code page (437) differs from Windows code page (1252)
8-bit characters might not work correctly. See psql reference
page "Notes for Windows users" for details.
You are now connected to database "postgres" as user "postgres".
postgres=# \c gisdb
WARNING: Console code page (437) differs from Windows code page (1252)
8-bit characters might not work correctly. See psql reference
page "Notes for Windows users" for details.
You are now connected to database "gisdb" as user "postgres".
gisdb=# _
```

We can create a database through the *psql* command as well:

gisdb=#create database testdb;

We can delete the database by the command:

gisdb=#drop database testdb;



```
Geog 676 Shell - psql -U postgres gisdb
gisdb=# create database testdb;
CREATE DATABASE
gisdb=# drop database testdb;
DROP DATABASE
gisdb=#
```

If you run the command \l to list all database again, you will find the database *testdb* is gone.

4. Create/Delete tables for your database

We will use three tables for this exercise including: country, city and river. The schema of the database is shown below:

Country (name: varchar (35), cont: varchar (35), pop: real, GDP: real, life_exp: real, shape: char(15))

City (name: varchar (35), country: varchar(35), pop: real, capital: char(1), shape: char (15))

River (name: varchar(35), origin: varchar(35), length: integer, shape: char(15))

In the country table, attribute name is country *name*, *cont* means the continent of the country, *GDP* represents gross domestic product, and *life_exp* is the life expectancy. The *shape* attribute indicates the geometry of a country, a city or a river.

The city table has five attributes: *name*, *country*, *pop*, *capital*, and *shape*. The country attribute is the name of the country that the city belongs to. Capital is a fixed character type, with length as 1, and two values (“Y” and “N”): a city will have a value “Y” for the attribute if it is the capital of the country, or “N” if it is not.

In the river, the *origin* attribute specifies the country where the river originates.

For each table, note that ***an underlined attribute is a primary key***. The concept of primary key will be introduced in the relational database design. Here, just keep in mind that it is used to uniquely identify each record in the table.

Question 2 (5 pts): What's the type of the schema presented here? (Please refer to the lecture 2 for the types of schema at different database design level).

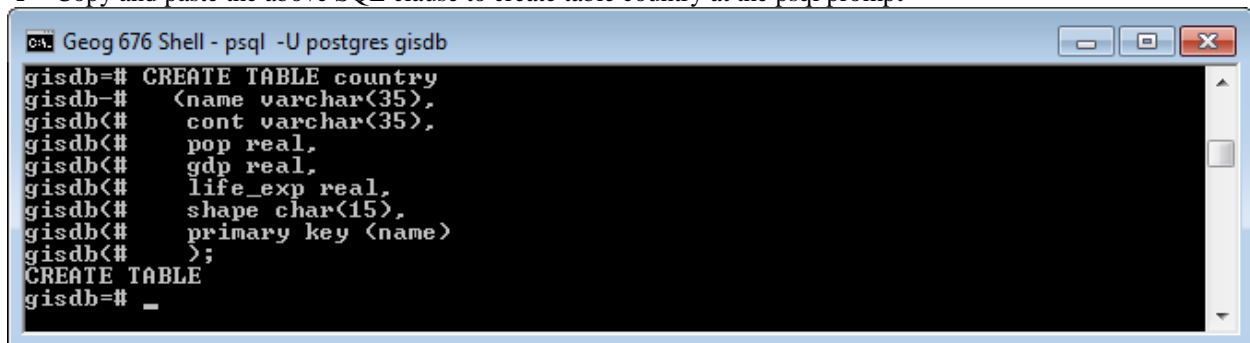
4.1 Create tables

In a relational database, the creation and deletion of the tables are specified in the DDL component of SQL. For example, the *country* schema introduced is defined below in SQL.

```
CREATE TABLE country(  
    name          varchar(35),  
    cont          varchar(35),  
    pop           real,  
    gdp           real,  
    life_exp      real,  
    shape         char(15),  
    primary key (name)  
);
```

The CREATE TABLE clause is used to define the relational schema. The name of the table is country.

→ Copy and paste the above SQL clause to create table country at the psql prompt



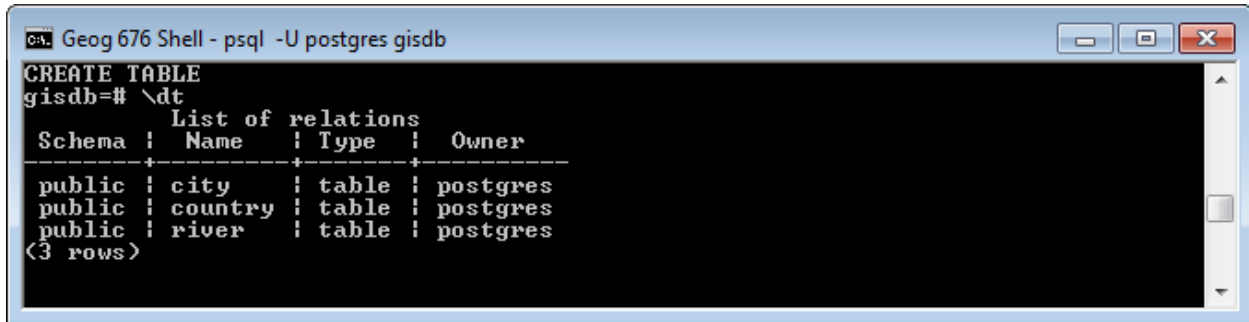
```
ca. Geog 676 Shell - psql -U postgres gisdb  
gisdb=# CREATE TABLE country  
gisdb=# (name varchar(35),  
gisdb=# cont varchar(35),  
gisdb=# pop real,  
gisdb=# gdp real,  
gisdb=# life_exp real,  
gisdb=# shape char(15),  
gisdb=# primary key (name)  
gisdb=# );  
CREATE TABLE  
gisdb=# _
```

→ Copy and paste the below SQL clause to create table river at the psql prompt

```
CREATE TABLE river(  
    name          varchar(35),  
    origin         varchar(35),  
    length         int,  
    shape         char(15),  
    primary key (name)  
);
```

Question 3 (5 pts): Following the DDL statements to create table country and river, create table city based on the city schema. Write down the DDL statements.

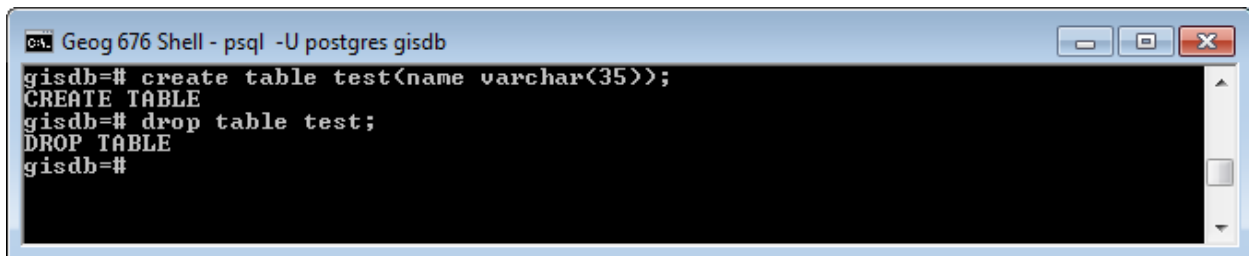
Now after you finished creating the three tables, you can use the command “\dt” to list all the tables.



```
ca. Geog 676 Shell - psql -U postgres gisdb
CREATE TABLE
gisdb=# \dt
          List of relations
Schema | Name      | Type  | Owner
-----+-----+-----+-----
public | city      | table | postgres
public | country   | table | postgres
public | river     | table | postgres
(3 rows)
```

4.2 Delete tables

Tables no longer in use can be removed from the database using **drop table** command.



```
ca. Geog 676 Shell - psql -U postgres gisdb
gisdb=# create table test(name varchar(35));
CREATE TABLE
gisdb=# drop table test;
DROP TABLE
gisdb=#
```

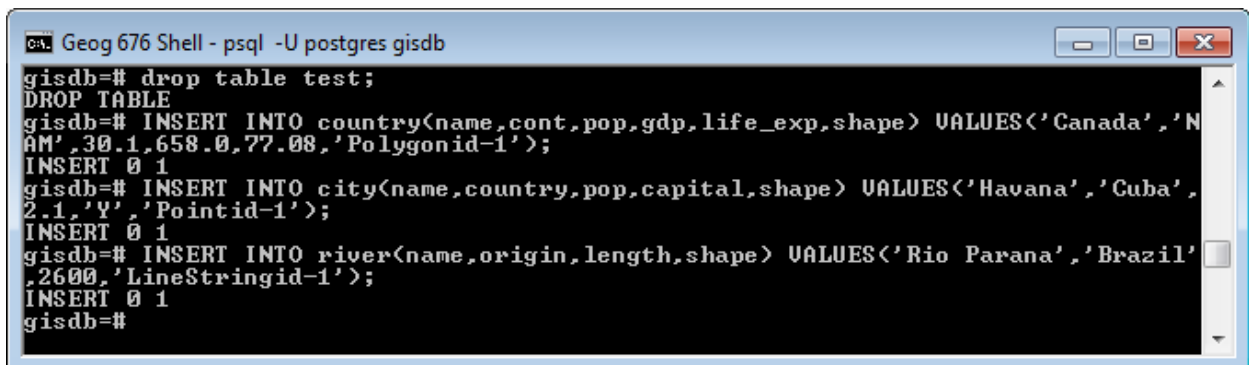
5. Add data to your database

After creating the tables as specified in DDL, it is time to add some data. This task, which is often called "populating the table", is done in the DML component of SQL. We can use SQL insert clause to create new rows for a table. For example, we can insert one row to table country, city and river respectively:

```
INSERT INTO country(name, cont, pop, gdp, life_exp, shape)
VALUES('Canada', 'NAM', 30.1, 658.0, 77.08, 'Polygonid-1');
```

```
INSERT INTO city(name, country, pop, capital, shape) VALUES('Havana', 'Cuba', 2.1, 'Y', 'Pointid-1');
```

```
INSERT INTO river(name, origin, length, shape) VALUES('Rio Parana', 'Brazil', 2600, 'LineStringid-1');
```



```
ca. Geog 676 Shell - psql -U postgres gisdb
gisdb=# drop table test;
DROP TABLE
gisdb=# INSERT INTO country(name,cont,pop,gdp,life_exp,shape) VALUES('Canada','NAM',30.1,658.0,77.08,'Polygonid-1');
INSERT 0 1
gisdb=# INSERT INTO city(name,country,pop,capital,shape) VALUES('Havana','Cuba',2.1,'Y','Pointid-1');
INSERT 0 1
gisdb=# INSERT INTO river(name,origin,length,shape) VALUES('Rio Parana','Brazil',2600,'LineStringid-1');
INSERT 0 1
gisdb=#
```

Note here if you add a row with a value for the attribute specified as the primary key already exists in the database, your statement will be rejected by the DBMS because of the primary key constraint specified in the DDL. This means that the value of the primary key attribute should be unique for each row, so it can

be uniquely identify each row in the table. For example, we insert another row in the river table with Name = "Rio Parana" again:

```
INSERT INTO river(name, origin, length, shape) VALUES('Rio Parana','Brazil',2500,'LineStringid-1');
```

The DBMS will generate and return errors to us as follows:

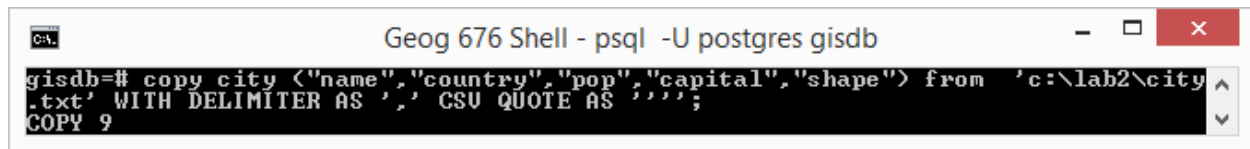
```
gisdb=# INSERT INTO river(name,origin,length,shape) VALUES('Rio Parana','Brazil',2500,'LineStringid-1');
ERROR: duplicate key value violates unique constraint "river_pkey"
DETAIL: Key (name)=(Rio Parana) already exists.
```

One fantastic way you may use frequently in future to load many data into the PostgreSQL is through a text file. This can be done with the command *"copy"*. The data read or written is a text file with one line per table row. Columns in a row are separated by the delimiter character. For example, we can use the following command to load the data from a text file, named as "city.txt", into the *city* table:

```
copy city ("name", "country", "pop", "capital", "shape") from 'c:\lab2\city.txt' WITH DELIMITER AS ',' CSV QUOTE AS ''';
```

or

```
copy city (name, country, pop, capital, shape) from 'c:\lab2\city.txt' WITH DELIMITER AS ',' CSV QUOTE AS ''';
```



Question 4 (8 pts): Now create a text file, following the similar format as the "city.txt" file, to keep the following rows for the country table, and use the copy command to load them to country table. Write down your copy command, and take a screenshot in pgAdmin III showing the resulted country table populated with records.

name	cont	pop	life_exp	gdp	shape
'Canada'	'NAM'	30.1	658	77.08	'Polygonid-1'
'Mexico'	'NAM'	107.5	694.3	69.36	'Polygonid-2'
'Brazil'	'SAM'	183.3	1004	65.6	'Polygonid-3'
'Cuba'	'NAM'	11.7	16.9	75.95	'Polygonid-4'
'USA'	'NAM'	270	8003	75.75	'Polygonid-5'
'Argentina'	'SAM'	36.3	348.2	70.75	'Polygonid-6'

6. Delete data from database

The basic form of SQL statement (DML) to remove rows from the table is as follows:

```
delete from table where <conditions>;
```

For example, the following statement removes the row from the table river that we insert above:

```
delete from river where name='Rio Parana';
```

¹<http://www.postgresql.org/docs/8.1/static/sql-copy.html>

```
Geog 676 Shell - psql -U postgres gisdb
gisdb=# delete from river where name='Rio Parana';
DELETE 1
gisdb=#
```

Question 5 (5 pts): Remove the row with country name "Mexico" from country table. Write down your command, and take a screenshot of your command line window.

7. Querying data with psql

Once the database schema has been defined in the DDL component and the tables populated, queries can be expressed in SQL to extract relevant subsets of data from the database. The basic syntax of an SQL query is extremely simple:

Select <column-names> from <table-names> where <conditions>;

Now let's list all the countries and continent they belong to in the country table by:

select name, cont from country;

```
gisdb=# select name, cont from country;
 name | cont
-----+-----
Canada | NAM
Mexico | NAM
Brazil | SAM
Cuba   | NAM
USA    | NAM
Argentina | SAM
(6 rows)
```

8. Running queries from an SQL file

We can use psql to run SQL code from an external file as well as from interactive input. This can be done with the `\i` psql command:

gisdb=#\i sqlfile

Now let's delete all records from the city table:

delete from city;

and load the data from SQL file "load-city.sql", which you downloaded from the Learn@UW system, and put under c:/lab2/:

gisdb=#\i c:/lab2/load-city.sql

```
gisdb=# delete from city;
DELETE 9
gisdb=# \i load-city.sql
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
gisdb=#
```

Question 6 (6 pts): List all the cities name and the country they belong to in the city table. Write down your SQL statement, and take a screenshot of the command line window showing your query results.

Under the same file directory, you will find another SQL file, named “load-river.sql”, which includes the insert statements for adding rows to the river table.

Question 7 (8 pts): Run “load-river.sql”, and list all rivers name and country where they originate. Write down your SQL statement, and take a screenshot of the command line window showing your query results.

Note that the sqlfile much under your current working directory, where you put your "c:/lab2". Otherwise, you must put absolute pathname for your SQL file. In PostgreSQL, an absolute pathname like C:\subdir\query.sql must be written either by doubling the backslashes, as in 'C:\\subdir\\query.sql', or turning them into forward slashes, as in 'C:/subdir/query.sql'.

Alternatively, you can run psql with the query file directly from the shell:

>psql -U dbuser -f sqlfile dbname

Now, let’s load more data to the country table by running a series of insert statement saved in the text file “load-counrty.sql” with the command:

>psql -U postgres -f load-country.sql gisdb

```
gisdb=# \q
C:\work\Dropbox\Geog676\lab\lab2>psql -U postgres -f load-country.sql gisdb
Password for user postgres:
psql:load-country.sql:1: ERROR:  duplicate key value violates unique constraint
"country_pkey"
```

Question 8 (8pts): Why PostgreSQL returns errors as above? Try to remove all rows in the country table and load the country data from the shell again. Write down your SQL statement to remove the rows, and take a screenshot of the command line window showing “load-counrty.sql” was successfully executed.

9. Backup/restore a PostgreSQL database

9.1 Backup a database

PostgreSQL provides the utility program *pg_dump* for database backup. The idea behind this dump method is to generate a text file with SQL commands that, when fed back to the server, will recreate the

database in the same state as it was at the time of the dump. To dump the database you can use the command:

```
>pg_dump -U postgres gisdb > gisdb.sql
```

where *gisdb* is the database name, and *gisdb.sql* is the output text file with SQL commands.

```
gisdb=# \q
C:\work\Dropbox\Geog676\lab\lab2>pg_dump -U postgres gisdb > gisdb.sql
Password:
C:\work\Dropbox\Geog676\lab\lab2>
```

Now you can go to your current working directory, to check the content of “gisdb.sql”.

10.1. Restore a database

The text files created by *pg_dump* are intended to be read in by the *psql* program. The general command form to restore a dump is

```
>psql -U username dbname < infile
```

```
C:\work\Dropbox\Geog676\lab\lab2>psql -U postgres gisdb < gisdb.sql
Password for user postgres: postgres
```

Part II: Lab Assignment

Answer Question 1 to Question 8 as specified in Part I: Exercise Tutorial. Write up your lab report and turn it in as a single .pdf file to Dropbox on Learn@UW.

Special tips: on Windows, **Alt + PrtScn** can help you get a screenshot of the current window.

Reference:

- PostgreSQL 9.3 Documentation (<http://www.postgresql.org/docs/9.3/static/index.html>)
 - II: [The SQL Language](#)
 - VI. Reference→ II. [PostgreSQL Client Applications](#)
- Shekhar, S., & Chawla, S. (2003). Spatial databases: a tour (Vol. 2003). Upper Saddle River, NJ: prentice hall.