INFO-445 Studio Workbook: Getting Started with PostgreSQL

For INFO-445/Spring 2013 we will be working with PostgreSQL 9.2.4 a very good database management system if not as popular as MySQL or other commercial systems:

http://www.postgresql.org/about/

PostgreSQL is open source. It is compliant with international standards. It has a rich set of data types. It is robust, highly extensible, quite fast, and very expressive. It has terrific documentation. And, whatever you learn about PostgreSQL is readily transferable to other databases including MySQL, Oracle, DB2, or Microsoft SQL Server.

Here's the documentation (don't print it – it's over 2,500 pages long):

http://www.postgresql.org/files/documentation/pdf/9.2/postgresql-9.2-US.pdf

And online:

http://www.postgresql.org/docs/9.2/static/index.html

This workbook is designed to get you started with PostgreSQL and basic web development.

Topics

A. How to install PostgreSQL on the UW student servers
B. How to use the PostgeSQL command line
C. How to work with a SQL script
D. How to create a simple PHP application for querying a database
E. Managing PostgreSQL: Using the pgAdmin III client
F. Front-end development: Using Microsoft Expression Web 4
G. PHP pgSql API: Very brief introduction
H. PL/pgSQL - SQL Procedural Language

Some Key Resources

PostgreSQL

Documentation
http://www.postgresql.org/docs/9.2/interactive/index.html

The command line
http://www.postgresql.org/docs/9.2/static/app-psql.html (copy & paste link)

SQL command reference
http://www.postgresql.org/docs/9.2/static/sql-commands.html (copy & past link)

PL/pgSQL - SQL Procedural Language
http://www.postgresql.org/docs/9.2/interactive/plpgsql.html

Other resources
http://www.postgresql.org/about/

Web Programming

Introduction
http://code.google.com/edu/submissions/uwspr2007_webprogramming/listing.html

PHP Language Reference
http://www.php.net/manual/en/langref.php

PostgreSQL API Reference
http://us3.php.net/manual/en/book.pgsql.php

Miscellaneous

Linux command line basics
http://linuxcommand.org/learning_the_shell.php

Source control – git
http://git-scm.com/
https://bitbucket.org/
http://sixrevisions.com/resources/git-tutorials-beginners/

Preliminaries

The Linux command line

I assume that you know the basics of the command line in Linux. If the Linux command line is new to you or if you are rusty you should review the basics here:

http://code.google.com/edu/tools101/linux/basics.html

You will also need to be able to edit text files on Linux. You can learn about the Pico editor here:

http://code.google.com/edu/tools101/linux/basics.html#create_new_file

Some reminders of some very simple Linux commands:

List the contents of a directory

> ls –las

Print the name of the current directory

> pwd

Go down into a directory (that is, go into public_html)

> cd public_html

Go up to the parent directory

> cd ..

Go to your home directory

> cd ~

Go to an absolute directory location – note those forward slashes (**/blah/blah**)

> cd /rc11/d99/i445g02/public_html

To create (or make) a new directory, called fred

> mkdir fred

To delete (or remove) a directory

> rmdir fred

Very Useful → The **Tab** and **Up Arrow** and **Down Arrow** keys

1. Use the Tab key to complete a file name or directory name
2. Use the Up Arrow and Down Arrow keys to move through previous commands

Basic web programming

I assume that you know basic web programming. You should, for example, feel comfortable with the material in this course:

http://code.google.com/edu/submissions/uwspr2007_webprogramming/listing.html

SSH: Secure Shell and File Transfer

- You will need a secure shell terminal and file transfer program -- Tera Term VT and FileZilla work well enough
- For more information on SSH see http://www.washington.edu/itconnect/web/publishing/ssh.html

A. How to install PostgreSQL on the UW student servers

You can install PostgreSQL on your UW student account, and use it with PHP or other apps directly from students.washington.edu.

(UW does not provide official directions for installing PostgreSQL. That said, the instructions for installing MySQL may be of some use: http://www.washington.edu/computing/web/publishing/mysql-install.html)

Comment: While the following steps for installing PostgreSQL are highly simplified and quite carefully specified, they give the flavor of what a Database Administrator might normally do on behalf of a software developer or development team.

Some students, because of prior experience, will find these steps almost trivial and will finish this installation in 20 minutes while watching TV. For other students, it may take several hours. If you get stuck stop, step aside, ask for some help, and then come back to it.

Database Administrators serve a highly technical role. Not only do they have to deal with a lot pressure – when things go wrong they come to the rescue – they also need to know a lot about the theory of databases and data access, about the specific design and implementation details of the installed database. And, they must be skilled system administrators and know a good deal about hardware and network configuration. To get a sense of some the tasks that are involved in administrating a PostgreSQL database you may want to briefly browse this Server Administration documentation

http://www.postgresql.org/docs/9.1/static/admin.html

To install PostgreSQL follow these steps:

1. Connect via SSH Tera Term to **vergil.u.washington.edu** using your **student NetID**. We will install PostgresSQL on this machine.

   A note: On your home directory of **dante** you will find a directory called **public_html** which is a symbolic link to file space, that is, a directory, on **vergil.u.washington.edu**. In this sense, vergil is a machine for web development and dante is a Unix machine for running databases and other software that you might install.

   Important: If you are using a **course NetID** then your Unix and web development environments are:
   homer.u.washington.edu
   ovid.u.washington.edu

   Machine names (for details see http://www.washington.edu/itconnect/web/publishing/urls-web.html)

	Unix	Web Dev	IP
vergil.u.washington.edu		x	140.142.0.0/16
dante.u.washington.edu	x		140.142.0.0/16
ovid.u.washington.edu		x	140.142.0.0/16
homer.u.washington.edu	x		140.142.0.0/16

   VPN Network

vpn.ischool.uw.edu		172.28.0.0/16

   On windows, for network details between two hosts use **pathping homer.u.washington.edu**

To install PostgreSQL follow these steps:

1. Connect with Connect via SSH Tera Term to **homer.u.washington.edu** using these credentials:

UW NetID:	i445s13**x**
Unix system:	homer.u.washington.edu
Web domain:	http://courses.washington.edu/i445s13**x**/
password:	*password for i445s13**x** to be given out*

   When you log in check for files and directories by typing
   ```
   > ls
   ```

   Check that the webserver is working by changing your directory to **public_html**
   ```
   > cd public_html
   ```

   Next edit the file **index.html** and save your changes.

   Check that your changes worked by viewing the page in the webserver:
   ```
   http://courses.washington.edu/i445s13x/
   ```

   Now, return to the root directory with this command
   ```
   > cd ~
   ```

2. Download PostgreSQL 9.1.3 source (mostly written in C), using this command
   ```
   > wget http://ftp.postgresql.org/pub/source/v9.2.4/postgresql-9.2.4.tar.gz
   ```

3. Un-compress the file using this command
   ```
   > tar xzvf postgresql-9.2.4.tar.gz
   ```

4. Navigate to the **postgresql-9.2.4** directory and build the source code. First you'll need to specify a directory for the PostgreSQL server. This should be a subdirectory of your home directory on the server.

   Use the **pwd** command to get the directory of your home server. In this example the user name is i445g02. For example
   ```
   > cd ~
   > pwd
   /rc11/d99/i445g02
   ```

   Now change your directory to
   ```
   > cd postgresql-9.2.4
   ```
   make

   If your home directory is **/rc11/d99/i445g02** and you want to install PostgreSql to the subdirectory named **postgresql**, you should enter the following command

   > ./configure --prefix=/rc11/d99/i445g02/postgresql --with-libxml --with-libxslt

   Comment: The flags --with-libxml and --with-libxslt are used so that PostgreSQL builds with support for XML.

Next, after the configure command is finished, enter the following command

```
> make && make install
```

*Comment: After this command finishes successfully, you have downloaded the source code (**wget**), uncompressed the source code (**tar xzvf**), configured the compiler with system-specific settings (**configure**), and setup the directory structures and compiled the source code (**make**).*

The PostgreSQL database management system (that is, the database server) is ready to run but before you can use PostgreSQL you need to complete several configuration steps. We turn to those next.

5. Now, you'll need to create a new data file (or database cluster) for your databases. Navigate to the directory that you just created: **postgresql**. Then, navigate to the **bin** subdirectory. You will find a number of database commands in this subdirectory. To list these commands type

```
> cd ~/postgresql/bin
> ls
```

Comment: If at some point you need to learn more about these commands consult the documentation. Each command is described in great detail there.

Use the **initdb** command to create a new database cluster. We recommend installing it into the **postgresql/data** directory. This is the command

```
> ./initdb ../data
```

6. You'll need to set a port on which the PostgreSQL database management system runs. This port will receive communication connections from your database clients, including programs that run in PHP, pgAdmin III (a GUI for managing PostgreSQL databases), and so forth. This port should be a number between 1024 and 65000 and must be a port that no one else is currently using. That is, you won't be able to use your friend's port number, for example – you will need your very own port number.

   In this step we will find a port for your server to use. Think of a random number between 1024 and 65000, and then type in this command, replacing **XXX** with the number you thought of:

```
> telnet localhost XXX
```

   (1) If it says "Connection Refused", then you have a **GOOD number. Write this number down and continue to the next step – note: you need to remember this port number.**

   (2) If it says something ending in "Connection closed by foreign host," then there is already a server running on that port, so you **should choose a DIFFERENT number and test it with the above telnet command.** (3) Repeat (1) until you get a GOOD port number.

7. Now, you need to edit the file **postgresql/data/postgresql.conf**. This is where you tell PostgreSQL to listen on the port that you just discovered. Uncomment the port line, and change the port number to your very own port number.

   You should look for the first line and change it to the second where **XXX** is the port number you found in the previous step (step #6)

```
#port = 5432                    # (change requires restart)
port = XXX                      # (change requires restart)
```

Note: The # sign is used to indicate that what follows is a comment. You must remove this comment.

Also uncomment the following and set the **listen_addresses** variable to a star ('*') as in

```
listen_addresses = '*'
```

8. You also need to edit the PostgreSQL client authentication configuration file, located at

```
~/postgresql/data/pg_hba.conf
```

pg_hba.conf is very important because it tells PostgreSQL who is allowed to connect to the server – not much works if authentication records are incorrectly specified.

Add the following two lines. The first line allows the UW webservers to connect to your database. This will allow your PHP scripts to make a connection with your database. The second line allows all users to connect to your database from any machine with password authentication.

```
# TYPE    DATABASE   USER        ADDRESS            METHOD
host      all        all         172.28.0.0/16      trust
hast      all        all         128.95.0.0/16      trust
```

This is often an effective permissions scheme – it allows the user xxx to make a connection to the database from any network:

```
hast      all        xxx         0.0.0.0/0            trust
```

Note: If you experience database connection problems it is probably because something is wrong with this file and the context in which you are trying to connect to the database.

~~*Note*: This may provide an additional level of security, allowing only hosts in the **ischool VPN network** to try to connect to the database.~~

```
#host      all        all         172.28.0.0/16   md5
```

9. You can now start the database. To start PostgreSQL, go to the **postgresql/bin** directory and enter

```
> ./pg_ctl start -D ../data
```

Note: Use this to record errors in a logfile

```
(> ./pg_ctl start -D ../data -l ../data/logfile)
```

Note: you may need to present the Enter key twice. If you get a message like this the database is running! It will continue running forever unless you tell it stop, an operator kills the process, the power fails, etc. Good work. You are almost there.

```
LOG:  database system was shut down at 2012-02-27 21:43:33 PST
LOG:  autovacuum launcher started
LOG:  database system is ready to accept connections
```

Note: If PostgreSQL complains about a port already in use, you'll need to change the port that PostgreSQL uses. In this case, go back to step #6 and step #7 and then try again.

To stop PostgreSQL, you can type the following. But, for now there is no need to stop your database

```
> ./pg_ctl stop -D ../data
```

To restart PostgreSQL, you can type the following:

```
> ./pg_ctl restart -D ../data
```

Note, further, that if you started PostgreSQL with **"-l ../data/logfile"** then the file **data/logfile** contains a log of server operations.

10. PostgreSQL provides a default database, named **postgres.** You can connect to this database using the **psql** command line tool. First, make sure that PostgreSQL is running. You check if it is running with this command

```
> ./pg_ctl status –D ../data
```

Now, go to the **postgresql/bin** directory and type the following (remember that **XXX** is the port number that you discovered in step #6)

```
> ./psql –p XXX -d postgres
```

You should see a command line for entering SQL commands! If so, good work – take a break.

```
psql (9.1.3)
Type "help" for help.

postgres=#
```

To leave the command line type

```
\q
```

11. You should create a PostgreSQL user ID and password. At the Unix command prompt in the PostgreSQL bin directory type the following command

```
./createuser -d –e -s -P -p XXX dave
```

These parameters indicate that databases can be created (**-d**), that the underlying SQL be echoed (**-e**), that a password should be prompted for (**-P**), that the connections should be made on the port **XXX (-p)**, and that the user is named **dave**.

```
Enter password for new role:
Enter it again:
Shall the new role be a superuser? (y/n) y
Shall the new role be allowed to create more new roles? (y/n) n
CREATE ROLE dave PASSWORD 'md5f1df1253ad33459069dc8da6d2ab4cd6' NOSUPERUSER
CREATEDB NOCREATEROLE INHERIT LOGIN;
```

Comment: For more about the create user command see the documentation here:
http://www.postgresql.org/docs/9.1/static/app-createuser.html

12. Start the **psql** and type the following command to alter your default db user name password

```
alter user your_user_id encrypted password 'pass';
```

Killing the server. Sometimes you need to be able to kill the server process. Here is how:

Use the following command to find the process ID of the postgreSQL server:
`> ps -ef | grep xxx` (Where xxx is the name of the login account)

If you need to delete a process use the following command:
`> kill -9 process_ID` (Where process_ID is the id of the process – see ps command)

Acknowledgements: Shaun Kane – a superb teaching assistant – originally developed these instructions in 2009. These instructions were revised substantially in 2012. Tony Grosinger helped improve these instructions in 2012.

B. How to use the PostgeSQL command line

The command line interface for interacting with the PostgreSQL database management system can be very efficient and very powerful. Often times you will use the command line to check something, to add a little data, to test the syntax for a comment, to make a quick change, or any number of other things. You can find help on the PostgeSQL command line here:
http://www.postgresql.org/docs/9.1/static/app-psql.html

It is often very helpful to know your command line. This said, like playing a musical instrument – or any kind of expert performance – it takes practice and time to learn. To get started, here is a very brief tour of a few things that you should know.

(1) Start the command line interface with this command (note your database needs to be running):

```
> ./psql –p XXX -d postgres      (XXX is the port on which your db is running)
```

(2) To get a list all the PostgreSQL SQL commands type

```
\h
```

(3) To leave the PostgreSQL command shell type

```
\q
```

(4) To get help with a particular SQL command type this kind of thing the command line will show you a syntax diagram for the command – often faster than going to the documentation

```
\h create schema

\h select
```

(5) To find out what tables (that is, relations) are in your database type

```
\d
```

(6) To list the system tables you can type

```
\dS
```

(7) The commands that begin with \ (backslash) are called meta-commands because they provide information or otherwise enable you to run SQL commands. To find a list of all the available shell commands type

```
\?
```

Notice all the commands that begin with \d , also known as the describe command. These various describe commands can be very useful for determining what is in your database.

(8) We will now turn to a small number of the other shell commands, mostly to give you a flavor of what can be achieved. You can see all the commands that you have recently typed into the shell with the following command

```
\s
```

(9) Even better you can write those commands to a file, called my_recent_cmds.txt, with

```
\w my_recent_cmds.txt
```

(10) You can also issue operating system commands from the PostgreSQL command shell. For example, to change your directory up one level you can type

```
\cd ..
```

(11) You can issue any operating system command with the \! command. For example, this sequence of three commands prints or current directory location, makes a directory, and changes to that new directory

```
\! pwd
\! mkdir working_data
\! cd working_data
```

(12) Now, let's do a little SQL. Type the following into the command line to create a schema called 'play' and a table called 'test.' Where is the table located? Where is the schema located? Note: SQL statements always end with a semi-colon (;).

```
CREATE SCHEMA play;
CREATE TABLE play.test (id SERIAL, name varchar(64), size integer);
```

(13) Now let's put two tuples into the table

```
INSERT INTO play.test (name, size) VALUES ('Bicycle', 100);
INSERT INTO play.test (name, size) VALUES ('Tricyle', 40);
```

(14) To query the table we can type in select statements such as

```
SELECT * from play.test;
SELECT * from play.test where name = 'Bicycle';
```

Rather than having to type commands over and over again you can press the "up arrow" to see the recently entered commands. Then, find the command that you need, modify it by moving the cursor and make edits, and then press enter. Try, for example, to issue some more INSERT commands.

(14) To determine what schemas are being used (also called name spaces) type

```
\dn
```

(15) To determine what's in a schema type

```
\dn play.*
```

(16) It is often very helpful to be able to copy the data from a relation to a text file. The copy command can be used to do that

```
\copy (select id, name, size from play.test) to my_test_table.txt
```

(17) This created a new text file called my_test_table.txt. Now, let's drop the relation play.test with this command

```
DROP TABLE play.test;
```

(18) With the text file we can re-create the table quite quickly with these two commands

```
CREATE TABLE play.test (id SERIAL, name varchar(64), size integer);
\copy play.test from my_test_table.txt
```

(19) You can confirm that the data is back into the table with this command

```
SELECT * from play.test;
```

(20) Let's delete all of our work. A clean way to delete everything is the following

```
DROP SCHEMA IF EXISTS play CASCADE;
```

(21) If this command seems odd check out the documentation – a good page to bookmark:
http://www.postgresql.org/docs/9.1/static/sql-commands.html

(22) It can be very useful to create a short SQL script and then execute the whole script. First, quit the command line interface with

```
\q
```

(23) Now, create a file called my_sql_script.sql with the following SQL lines

```
/* filename: my_sql_script.sql */

/*
  * Create blank schema for holding this simple example
  */
 DROP SCHEMA IF EXISTS play CASCADE;
CREATE SCHEMA play;

/*
  * Create a simple table
  */
CREATE TABLE play.test (
   Id       SERIAL,
   name     varchar(64),
   size     integer);

/*
      * Load some data into the table
      */
INSERT INTO play.test (name, size) VALUES ('Bicycle', 100);
INSERT INTO play.test (name, size) VALUES ('Tricyle', 40);

/*
     *  Show that the table can be queried
     */
SELECT * from play.test;
SELECT count(*) from play.test;
```

(24) After you've created this script you can read the input from a file and it will execute as if it had been typed on the keyboard. First, start the command line interface with

```
> ./psql –p XXX -d postgres
```

(25) Then type in this command

```
\i my_sql_script.sql
```

(26) One final approach that is often quite helpful. From the Linux command line you can type the following to have SQL script executed

```
./psql -p 4567 -d postgres < my_sql_script.sql
```

This says run a psql program on port # 4567 with the database named postgres (a default database which is created automatically for you) and redirect the SQL statements that are found in the file my_sql_script.sql as input. The output of the SQL statements will be presented on the terminal. To have the output redirected into a file, here called output.txt, use the command (*Note*: be sure to chance the port 4567 to your port):

```
./psql -p 4567 -d postgres < my_sql_script.sql > output.txt
```

C. How to work with a SQL script

Consider the following database application, which creates a schema for holding the application, creates a single relation, defines a function with PL/SQL, and inserts four tuples into the relation. Go ahead and run this code in PostgeSQL.

You can find the code here:
http://courses.washington.edu/i445g02/INFO-445/front_door.html

Once the SQL script below has been executed by PostgreSQL, the relation can be queried and updated with new data. For example we could enter the following queries:

```
select * from play.snack;
select * from play.howManySnacks();
```

The SQL script:

```
/* filename: snacks.sql */

/*
 * The system that we will be using plpgsql scripts
 */
create or replace language plpgsql;

/*
 * Create blank schema for holding this simple example
 */
DROP SCHEMA IF EXISTS play CASCADE;
CREATE SCHEMA play;

/*
 * Create a simple table
 */
CREATE TABLE play.snack (
      id          SERIAL PRIMARY KEY,
      name        varchar(64) UNIQUE NOT NULL,
      calories    integer CHECK (calories > 0),
      price       numeric(8,2)
);

/*
 * Returns the number of snacks in the database
 */
CREATE OR REPLACE FUNCTION play.howManySnacks ()
RETURNS INTEGER AS '
DECLARE
       num     integer;
BEGIN
       select into num count(*) from snacks;
       return num;
END;
' LANGUAGE plpgsql;

/*
 * Load some data into the table
 */
INSERT INTO play.snack (name, calories, price) VALUES ('Chips', 200, 3.50);
INSERT INTO play.snack (name, calories, price) VALUES ('Cheezees', 200, 3.50);
INSERT INTO play.snack (name, calories, price) VALUES ('Bar', 300, 2.00);
INSERT INTO play.snack (name, calories, price) VALUES ('Fries', 500, 4.25);
```

D. How to create a simple PHP application for querying a database

Once you have executed the snacks.sql script and the snack relation has been created in your database you are ready to create a PHP application. The PHP application will consist of a small number of files, including:

font_door.html
DBVars.php
show_snaks.php

In your web development directory create a directory called **snack_app**

```
> mkdir snack_app
```

You will access your application in your browser with the following http request

```
http://courses.washington.edu/i445g02/snacks_app/show_snacks.php
```

The first file, **front_door.html**, is very simple

```
<html>
   <head>
      <title>Snacks: Front door</title>
   </head>
   <body>
      <h3>Snacks application</h3>
      <p>
      <ul>
      <li> <a href="show_snacks.php">Show snacks</a>
      </ul>
   </body>
</html>
```

It simply shows a web page with a link to a PHP script.

The **DBVars.php** script looks like this. This script simply creates some global variables that are used in other scripts. You will need to carefully edit this script so that it is configured for your database server.

```
<?php
/*
 *filename: DBVars.php
 */

$gDB_host   = "homer.u.washington.edu";
$gDB_name   = 'postgres';
$gDB_port     = '45673';
$gDB_user   = 'i445g02';
$gDB_password = "pass";
$gDB_conn_string =  'host=' .      $gDB_host .
                    ' dbname=' .   $gDB_name .
                    ' port=' .     $gDB_port .
                    ' user=' .     $gDB_user .
                    ' password=' .   $gDB_password .
                    '';
?>
```

The second PHP script, **show_snacks.php**, does the interesting pieces. This script makes a connection to your database, as configured in DBVars.php. Then, it creates a simple SQL query. Then, the query is run. The results are then presented. Finally, the script cleans up the results and closes the connection.

```
<html>
   <head>
      <title>Snacks: ProstgreSQL Database Test</title>
   </head>
   <body>
      <?php
         // include key database variables for connection string
         include 'DBVars.php';
         echo "\nConnection string >" . $gDB_conn_string . "<" . "\n<p>\n";

         // Try to make a connection
         $db = pg_connect($gDB_conn_string);
         if (!$db) {
            die("Error in connection: " . pg_last_error());
         }

         // Create and run a query
         $sql = "SELECT * FROM play.snack";
         echo "The SQL query >" . $sql . "<\n<p>\n";
         $result = pg_query($db, $sql);
         if (!$result) {
            die("Error in SQL query: " . pg_last_error());
         }

         // Show some snacks
         while ($row = pg_fetch_array($result)) {
            echo "Snack " . $row[0] . " (" . $row[1] . ") has " . $row[2] . " calories!
<br/>\n";
         }

         // wrap up
         pg_free_result($result);
         pg_close($db);
      ?>
   </body>
</html>
```

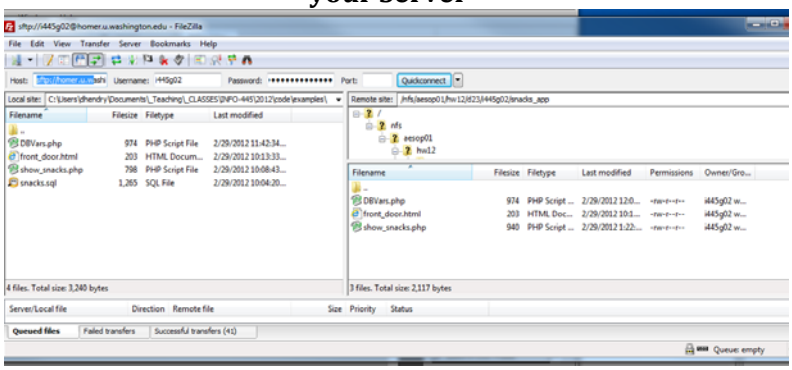
After the script runs it generates HTML that will look something like this

```
<html>
   <head>
      <title>Snacks: ProstgreSQL Database Test</title>
   </head>
   <body>

Connection string >host=homer.u.washington.edu dbname=postgres port=45673 user=i445g02<
<p>
The SQL query >SELECT * FROM play.snack<
<p>
Snack 1 (Chips) has 200 calories!<br/>
Snack 2 (Cheezees) has 200 calories!<br/>
Snack 3 (Bar) has 300 calories!<br/>
Snack 4 (Fries) has 500 calories!<br/>
Snack 5 (Popcorn) has 200 calories!<br/>
Snack 6 (Pop) has 400 calories!<br/>

   </body>
</html>
```

A very simple approach for implementing this application is to use Notepad++ as your editor. When you are happy your code use FileZilla to copy your files from the local development directory to the web server.

This development strategy is simple but slow. We will introduce a more robust solution below.

FileZilla for Copying Files to your Server



Notepad++ for Editing Files

E. Managing PostgreSQL: How to use pgAdmin III

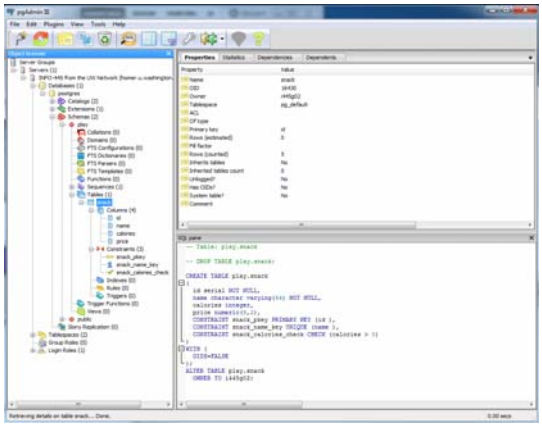
pgAdmin II is a client application that runs on your desktop computer and allows you to control your PostgreSQL database server. It is often much easier to use a client application for controlling the server. Here is a 10 minute video on its use

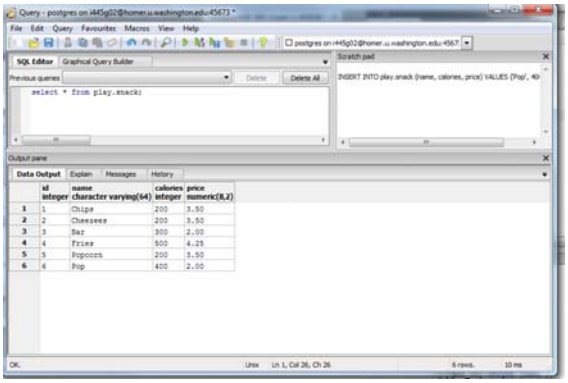
http://www.enterprisedb.com/resources-community/webcasts-podcasts-videos/videos/how-create-postgres-database-using-pgadmin

The pgAdmin II application can be downloaded here

http://www.pgadmin.org/index.php

You should install this application on your development machine – you will find it to invaluable.





If you run into connection errors it is probably because of a misconfiguration in

~/postgresql/data/pg_hba.conf

Debugging this configuration file can be difficult. Here are some steps that you can take:

1. Work together with other students and debug each other's setups;
2. Confirm that PostgreSQL is running;
3. Confirm that you can access PostgreSQL from the command line with psql;
4. Confirm that you have a user role defined and a password for that user role --
   See CREATE USER … and ALTER USER …
5. Open pg_hba.conf and careful inspect each line and confirm that you have the correct access records:

```
# TYPE    DATABASE   USER        ADDRESS          METHOD
host      all         all         140.142.0.0/16     trust
host      all         all         0.0.0.0/0          md5
```

6. Make sure that the connection properties in pgAdmin III are correct;
7. Make sure that you've connected through the iSchool VPN (or that the network address is otherwise correctly stated).

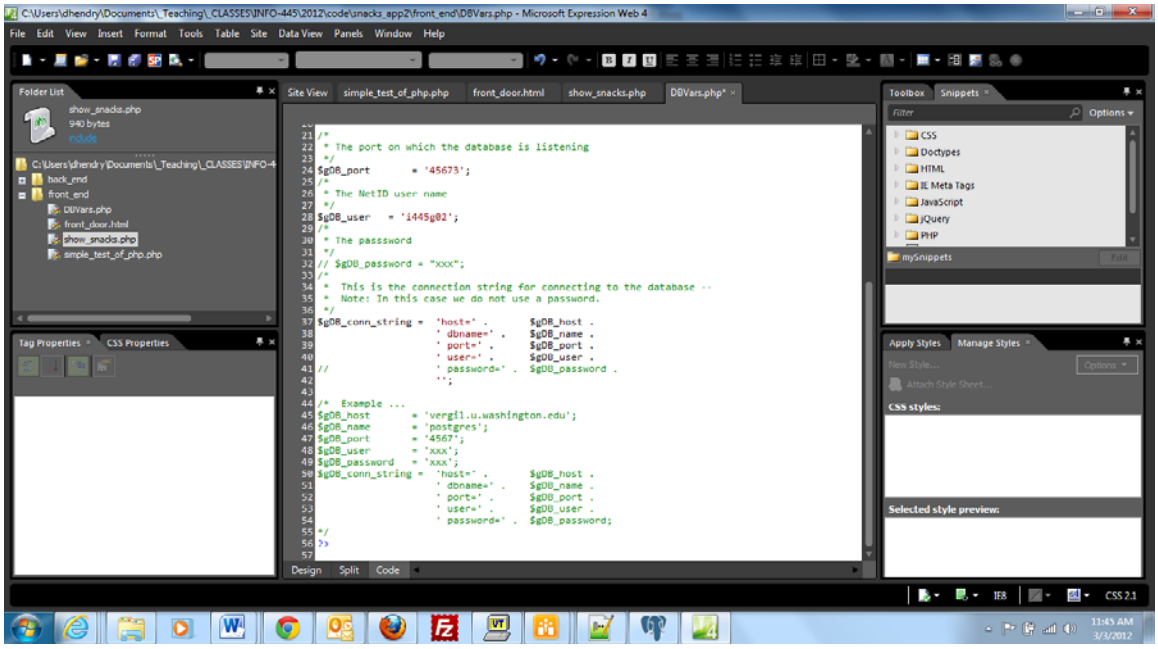
Incidentally, a good number of PostgreSQL GUI tools are available. See, for example:

http://wiki.postgresql.org/wiki/Community_Guide_to_PostgreSQL_GUI_Tools

F. Front-end development: Using Microsoft Expression Web 4

In INFO-445 you will need to create basic user-interfaces to test and demonstrated your back-end database design work. Many development tools exist for this purpose including Eclipse, NetBeans, Dreamweaver, Visual Studio and so forth.

For INFO-445 we'll be using Microsoft Web Expressions which appears to be a good, relatively simple tool for creating front-end code. You are likely to find it to be far more efficient to use than NotePad++.



Before you start using Web Expressions to create front-end applications you will need to set up some things.

(1) Find the location of **notepad++.exe** and write down the path to this executable. We will use this editor for editing SQL scripts. On my machine the location is "C:\Program Files (x86)\Notepad++\notepad++.exe"

(2) Download the PHP binary (PHP 5.4) for Windows from this site: http://windows.php.net/download/

a. Place the file containing the binary in a directory. Unzip the file.
b. Locate the file php.ini-development
c. Copy and rename this file to php.ini
d. Edit php.ini and make sure that
    i. The **extension_dir** variable is set to **extension_dir = "ext"**
    ii. The following two extensions are uncommented

```
extension=php_pdo_pgsql.dll
extension=php_pgsql.dll
```

(3) Download the snacks_app.zip file from the course website and unzip it into a working directory. When you unzip it the directory structure will look something like this:

```
…\info-445\code\snacks_app\
…\info-445\code\snacks_app\back_end
…\info-445\code\snacks_app\back_end\snacks.sql
…\info-445\code\snacks_app\front_end\DBVars.php
…\info-445\code\snacks_app\front_end\front_door.html
…\info-445\code\snacks_app\front_end\show_snacks.php
…\info-445\code\snacks_app\front_end\simple_test_of_php.php
```

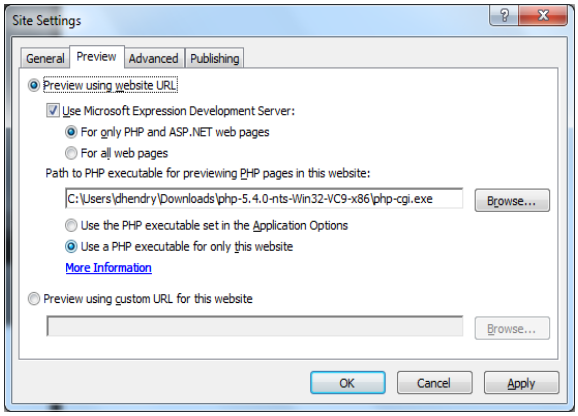
(4) Now, we are ready to do the setup. First, we need to create a site for the code. Follow these steps:

(1) Open Web Expressions 4
(2) On the top menu, click on Site > New Site …
(3) Select Empty Site and select a directory location for your site
(4) Add the back_end and front_end folders to that location.

You should now see those folders and files in the Folder List panel on the left.

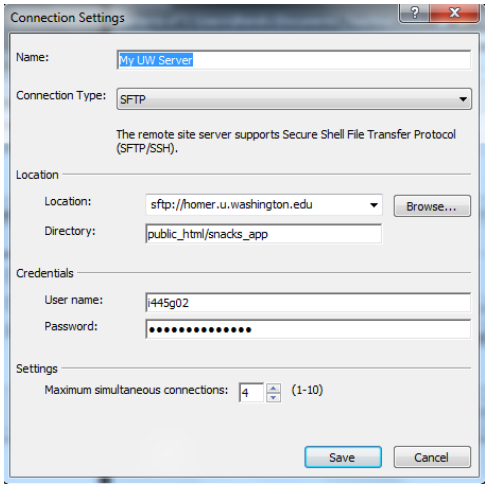
(5) Next, we assign a file editor to files with the SQL extension. To do this, select the **back_end** folder and right click on the **snacks.sql** file. Change the open program to Notepad++. (You found the path earlier, in one of the above steps.)

(6) Next, we will set up Web Expressions so that we can preview PHP files. This step will greatly reduce your development effort because you want have to move files up to your server every time you need to debug your code. On the top menu, click on **Site > Site Settings…** and then click on the Preview tab. Click the radio buttons as you see below and give the path to the file **php-cgi.exe**. (You downloaded the PHP binary earlier.)



Once you set up the path to the php-cgi.exe you can preview PHP pages in the browser. (Note that Web Expressions allows you to easily compare page across web pages.)

(7) Next, we will set up Web Expressions so that your site can be easily published to the server machine. To do this click on the **Publishing** tab. Then, click on the **Add...** button. As you can see in this dialog, you need to enter a name for your connection, select the Secure Shell File Transfer Protocol, your location, and credentials:



Once you've set up this settings you can select the menu item **Publish All Files to "My UW Server"** and similar publishing items.

G. PHP pgSql API: Very brief introduction

The follow pgSQL API code examples comes from
http://us3.php.net/manual/en/book.pgsql.php

You should feel comfortable with each of these functions and the code snippets below.

pg_connect – Used to create a connection with the database
http://us3.php.net/manual/en/function.pg-connect.php

```
$dbconn3 = pg_connect("host=sheep port=5432 dbname=mary user=lamb password=foo");
```

pg_query – Used to execute a query to a database
http://us3.php.net/manual/en/function.pg-query.php

```
$conn = pg_pconnect("dbname=publisher");
if (!$conn) {
echo "An error occured.\n";
echo
exit;
}

$result = pg_query($conn, "SELECT author, email FROM authors");
if (!$result) {
echo "An error occured.\n";
echo pg_last_error($conn);
exit;
}

while ($row = pg_fetch_row($result)) {
echo "Author: $row[0] E-mail: $row[1]";
echo "<br />\n";
}

<?php

$conn = pg_pconnect("dbname=publisher");

// these statements will be executed as one transaction

$query = "UPDATE authors SET author=UPPER(author) WHERE id=1;";
$query .= "UPDATE authors SET author=LOWER(author) WHERE id=2;";
$query .= "UPDATE authors SET author=NULL WHERE id=3;";

pg_query($conn, $query);

?>
```

pg_escape_string – This correct escapes strings so that they can be stored in tables
http://us3.php.net/manual/en/function.pg-escape-string.php

```
<?php
// Connect to the database
$dbconn = pg_connect('dbname=foo');

// Read in a text file (containing apostrophes and backslashes)
$data = file_get_contents('letter.txt');

// Escape the text data
$escaped = pg_escape_string($data);

// Insert it into the database
pg_query("INSERT INTO correspondence (name, data) VALUES ('My letter',
'{$escaped}')");
?>
```

pg_fetch_row – Used to access a row from a result set
http://us3.php.net/manual/en/function.pg-fetch-array.php

```
<?php

$conn = pg_pconnect("dbname=publisher");
if (!$conn) {
echo "An error occured.\n";
exit;
}

$result = pg_query($conn, "SELECT author, email FROM authors");
if (!$result) {
echo "An error occured.\n";
exit;
}

$arr = pg_fetch_array($result, 0, PGSQL_NUM);
echo $arr[0] . " <- Row 1 Author\n";
echo $arr[1] . " <- Row 1 E-mail\n";

// As of PHP 4.1.0, the row parameter is optional; NULL can be passed instead,
// to pass a result_type. Successive calls to pg_fetch_array will return the
// next row.
$arr = pg_fetch_array($result, NULL, PGSQL_ASSOC);
echo $arr["author"] . " <- Row 2 Author\n";
echo $arr["email"] . " <- Row 2 E-mail\n";

$arr = pg_fetch_array($result);
echo $arr["author"] . " <- Row 3 Author\n";
echo $arr[1] . " <- Row 3 E-mail\n";

?>
```

pg_fetch_object – Used to create a PHP object out of a tuple
http://us3.php.net/manual/en/function.pg-fetch-object.php

```
<?php

$database = "store";

$db_conn = pg_connect("host=localhost port=5432 dbname=$database");
if (!$db_conn) {
```

```
echo "Failed connecting to postgres database $database\n";
exit;
}

$qu = pg_query($db_conn, "SELECT * FROM books ORDER BY author");


while ($data = pg_fetch_object($qu)) {
echo $data->author . " (";
echo $data->year . "): ";
echo $data->title . "<br />";
}

pg_free_result($qu);
pg_close($db_conn);

?>
```

E. PL/pgSQL - SQL Procedural Language

PostgreSQL provides a way for extending SQL with application-specific functions. PL/pgSQL is one language that can be used to write these functions (other languages include Tcl, Perl, and Python).

PL/pgSQL provides a fairly rich set of control structures (**`if … then, case, while … end loop, for … in … loop`**, etc.) and language constructs for querying and updating databases. For details, see http://www.postgresql.org/docs/9.1/interactive/plpgsql.html

Example 1

```
1   /*
2    * Add a device type
3    */
4   CREATE OR REPLACE FUNCTION uwm.Add_device_type (
5      p_id         int,  // the device ID
6      p_descr      text  // the description of the device
7   )
8   RETURNS void AS '
9
10  DECLARE
11     row RECORD;
12
13  BEGIN
14    select into row * from uwm.device_type where p_id=id;
15    IF NOT FOUND THEN
16       INSERT  INTO uwm.device_type (id, descr)
17               VALUES ( p_id, p_descr);
18    END IF;
19  END;
20  ' LANGUAGE plpgsql;
```

Shown above is function written in PL/pgSQL. Functions are declared with the statement **`create or replace function`** and a list of parameters. Parameters are named and typed. Any type in PostgreSQL can be used. Lines 5-6 show two parameters, an integer and a text.

Line 8 says that the function returns **`void`**, that is, it does not return any value. The body of the function is written within a string. In the **`DECLARE`** section you declare the variables that you will be using in the function. Unlike PHP and other scripting languages, you *must* declare all variables and their types in this section. Line 13 indicates that the code for the function is to begin and the procedure continues until the final **`end`** (line 19). The last line indicates that the string that makes up the procedure should be treated as **`plpgsql`**. Line 14 performs a query and puts the results into the variable **`row`**. If the **`select`** statement does not find a record then an **`insert`** statement is executed and the procedure finishes; otherwise, the **`insert`** statement is *not* executed and the procedure finishes.

Example 2

This example shows the construction of a query in a string, then executing the string, and returning a set of tuples.

```
1   CREATE OR REPLACE FUNCTION dtw.Find_docs (
1     p_doc_type      dtw.doc_type_type,
2     p_offset     int,
3     p_limit         int
4     )
5   RETURNS SETOF dtw.doc as $PROC$
6   DECLARE
7     row1   dtw.doc%ROWTYPE;
8     q      text;
9     w      text;
10  BEGIN
11    if (p_doc_type = 'Z' or p_doc_type = NULL) then
12       w := ' ';
13    else
14       w := 'where status = ' || '' || p_doc_type || '';
15    end if;
16
17    if (p_limit = -1) then
18       q := 'select * from dtw.doc ' || w || ' order by entered desc limit ALL offset $1';
19       FOR row1 IN EXECUTE q USING p_offset
20          LOOP
21             return next row1;
22          END LOOP;
23       RETURN;
24    else
25       q := 'select * from dtw.doc ' || w || ' order by entered desc limit $1 offset $2';
26       FOR row1 IN EXECUTE q USING p_limit, p_offset
27          LOOP
28             return next row1;
29          END LOOP;
30       RETURN;
31    end if;
32  END;
33  $PROC$ LANGUAGE plpgsql;
```

Example 3

This is a more complex example that illustrates some of the expressive power and additional control structures of **plpgsql.**

```
34  /*
35   * Use this function to return the most recent readings
36   */
37  CREATE OR REPLACE FUNCTION uwm.get_new_readings (
38     p_did          int,
39     p_delta     char(1),
40     p_max       int
41  )
42  RETURNS SETOF uwm.meter_reading as $PROC$
43  DECLARE
44    num          int;
45    row          uwm.meter_reading%ROWTYPE;
46    row2         RECORD;
47    q            text;
48    time_cond    text;
49    MAX          int;
50
51  BEGIN
52  /*
53   * Check that the did is valid before trying to query
54   */
55  SELECT into row2 D.id as did from uwm.device D where D.id=p_did;
56  IF NOT FOUND THEN
57      perform uwm.Post_Message('Get_Readings', '',
58      'ERROR-Invalid did-' || p_did, p_did,NULL);
59      RETURN;
60  END IF;
61
62  /*
63   * Check input parameters
64   */
65  MAX := 3;
66  if p_max > 0 then
67      MAX := p_max;
68  end if;
69
70  if (p_delta = 'A') then
71       time_cond := ' AND 1 = 1 ';
72  elseif (p_delta = 'B') then
73       time_cond := ' AND L.timelogged < ((select max(timelogged) from uwm.work_log where '
74                      || 'did=' || p_did || ')) ';
75  elseif (p_delta = 'C') then
76        time_cond :=    ' AND L.timelogged < ((select max(timelogged) from uwm.work_log
77             where ' || 'did=' || p_did || ') - interval ''1 day'') ';
78  elseif (p_delta = 'D') then
79        time_cond := ' AND L.timelogged < ((select max(timelogged) from uwm.work_log where '
80            || 'did=' || p_did || ') - interval ''1 week'') ';
81  elseif (p_delta = 'E') then
82         time_cond :=  ' AND L.timelogged < ((select max(timelogged) from uwm.work_log
83             where ' || 'did=' || p_did || ') - interval ''1 month'') ';
84  elseif (p_delta = 'F') then
85        time_cond :=    ' AND L.timelogged < ((select max(timelogged) from uwm.work_log
86             where ' ||  'did=' || p_did || ') - interval ''1 year'') ';
87  end if;
88
89  q := 'select L.did as did, L.timelogged as t0, L.e_reading as er, L.e as e, ';
90  q := q || 'L.interpolated as I, L.year as y, L.moy as moy, L.woy as woy, L.dow as dow';
91  q := q || ' from uwm.work_log L ';
92  q := q || ' where L.did=' || p_did ;
93  q := q || time_cond;
94  q := q || ' order by L.timelogged desc';
95
96  /*
```

```
97   * Number of records read
98   */
99  num := 0;
100
101 FOR row2 IN execute q
102    LOOP
103       num := num + 1;
104       if (MAX <> 0) then
105          if (num = MAX + 1) then
106             RETURN;
107          end if;
108       end if;
109
110       row.did         := row2.did;
111       row.t0          := row2.t0;
112       row.e_reading   := row2.er;
113       row.e           := row2.e;
114       row.interpol    := row2.I;
115       row.nulls       := 0;
116
117       row.year := row2.y;
118       row.woy  := row2.woy;
119       row.moy  := row2.moy;
120       row.dow  := row2.dow;
121       row.info := '';
122       RETURN NEXT row;
123    END LOOP;
124    RETURN;
125 END;
126 $PROC$ LANGUAGE plpgsql;
127
128 /*
129  *  Used as a record set format for returning data -- see get_readings
130  */
131 drop table if exists uwm.meter_reading cascade;
132 create table uwm.meter_reading (
133      did          int,
134      t0           timestamp with time zone,
135      t1           timestamp with time zone,
136      e_reading    numeric(16,2),
137      e           real,
138      interpol     int,
139      nulls        int,
140      dow          int,
141      moy          int,
142      year         int,
143      woy          int,
144      info        char(16)
145 );
```