Exercise Sheet 0

Question 1: Using PostgreSQL

a. Log into the UNIX system.

b. For this module, we will be using the database server callled

```
mod-fund-databases.cs.bham.ac.uk
```

By the way, it is a separate machine in the School's network, running the Linux operating system.

c. Your initial password is stored under your home directory in the file

```
~/work/db-passwords/fundamentals-password.txt
```

(If you don't have such a file, then you need to see the IT support staff.)

d. Start up the database client program by running the command

```
psql -h mod-fund-databases fundamentals
```

The `-h` option tells `psql` that the database host is called `mod-fund-databases`. You can omit the `.cs.bham.ac.uk` part while you are inside the School network. The last part asks to use the `fundamentals` database that has been created for your exercises.

e. `psql` will ask you for a password. Type in the password you found in your file. You can cut and paste it so that you don't make any typing mistakes.

f. You should now see the `psql` command prompt, which might look like:

```
fundamentals=>
```

If you did not put `fundamentals` as the name of the database, then you are likely to get an error, because PostgreSQL looks for a database with the same name as your login in, which doesn't exist (yet).

g. There are two kinds of commands, SQL queries and PostgreSQL-specific "house-keeping" commands. The course is intended to teach you about the former and we will try some simple queries later on this exercise sheet. To start with, use an SQL command to change your password to something you can actually remember (note the semicolon at the end of the command):

```
ALTER USER mscNNxxx WITH PASSWORD 'xxxxxxxx';
```

Do not re-use your UNIX password for connecting to the database! The way passwords are stored by PostgreSQL is *not* as secure as it should be. On the other hand, you *can* use something that you can easily remember such as your great-great-great grandfather's middle name or your high-school sweetheart's favourite dissert or something, because somebody who finds out about your PostgreSQL password will not be able to do a lot of damage.

h. Log out of PostgreSQL by typing `\q` and log back into it, using the new password. Hopefully, it has all worked!

i. The PostgreSQL house-keeping commands are very useful. You get a listing of all of them by issuing

```
\?
```

For example, try the following

```
\dt
```

which will list all tables available to you in the currently selected database. The command

```
\l
```

will list all databases on the system. There are many but in most cases you are not allowed to read what is in them, unless the owner has granted you read (or write) access explicitly. One database that you can all read is called `fundamentals`, which I have populated with a few tables for the purposes of this course. Connect to it by issuing

```
\c fundamentals
```

and then issue `\dt` again to see the names of the tables I have created (there are seven).

j. In order to see the contents of a particular table, use SQL. Type

```
SELECT * FROM staff;
```

to see information about members of this department. Since this is SQL, don't forget to type the semicolon at the end of the query. If you don't, `psql` will just wait for more input. What this means, in particular, is that you can type long queries on several lines.

`psql` will now display the contents of the table `staff` using the program that is prescribed by your own personal shell variable `PAGER`. By default, the system administrators set the variable to point to the program `more` that you may know already. It is normally used to display the contents of UNIX files. `more` displays the table page by page. You go from one page to the next by pressing the space bar.

k. Personally, I find it useful to also be able to go back to previous pages but `more` does not allow one to do this. For this purpose there is the more powerful (but somewhat slower) program `less`. I suggest that you switch to it as well, by logging out of `psql` now (using `\q` again), and issuing the following command on the UNIX command line

```
setenv PAGER less
```

You should also include this line in your `.login` file so that it is set every time you log in. Do this now.

In `less` you can go back a page by typing "`u`" (for "up"). You tell `less` that you are finished with looking at a file (or a table, in our case), by typing "`q`" (for "quit").

l. What is the `less` command for going back to the beginning of a file? Find out by typing "`h`" (for "help") while `less` displays a table. Write down the answer on a sheet of paper.

m. Look at all the tables by appropriately changing the SQL command above. How many entries (also called "rows" or "records") does each of them contain. (Don't count yourself!)

n. What are the attribute names in the `staff` table? Do you understand their meaning?

o. The command line editor of `psql` is very comfortable. You can bring up previous commands by typing "`Ctrl-p`" (for "previous"). If, by accident, you went back too far, then type "`Ctrl-n`" (for "next"). Once you have a previous command, you can navigate in it by using the two cursor keys $\rightarrow$ and $\leftarrow$, and use "`Ctrl-d`" (for "delete") or "`backspace`" for rubbing out the letters that need changing.

p. Try out the on-line editing features!

q. SQL is not very fussy about how you write keywords or table names; you can use any combination of lower and upper case letters as you like, but note that the spelling must be right. Try changing the case of characters in the queries issued before.

r. Now develop some simple SQL queries:

   i. Who occupies office number 211?

ii. Where is the office of Dr Mark Ryan? (NB: This requires to test for equality of a string. Remember that strings are enclosed in single quotes, and that equality only holds if the two strings are equal character by character. Check this statement by adding some blanks at the end of the string `'Ryan'` in your query or by changing the case of a letter.)

iii. Which staff have offices on the first floor?

iv. Who are the professors in the School?

s. Leave the program `psql` by typing `\q`.

Question 2: More PostgreSQL

a. It is tedious to develop a complex SQL query interactively via the client program `psql`. A much better approach is to write the query into a file and then to load that file into `psql` for execution. Let's try this mechanism first.

i. Log into the UNIX system in the usual way. You should get at least one command-line window.

ii. Open your favourite editor (say, `emacs` or `nedit`) in a separate window. Resize the windows so that they don't overlap, ideally horizontally above each other and extended to full screen width.

iii. In the command-line window, create a directory called "`db`" by issuing the command

```
mkdir db
```

Change into the new directory by issuing

```
cd db
```

iv. In the editor window, open a new file in directory "`db`", let's call it "`q1.sql`" (short for "query 1"). Enter a simple query into the file, such as

```
SELECT * FROM staff;
```

Now save the file to disk (but don't exit the editor).

v. In the command-line window, start up the database client program:

```
psql -h mod-fund-databases fundamentals
```

Rather than typing in a query, you can load a file containing a query. Let's try this with the file we just created:

```
\i q1.sql
```

If you get the response "`q1.sql: No such file or directory`" then query file and `psql` weren't in the same directory. These notes are written with the assumption that we are in the working directory "`db`".

vi. You can now go back to the editor window, change the query to something more interesting, save the file, and load again from within `psql`. The editor gives you a much better overview of the query (particularly if you follow some layout conventions) than the command line of `psql` can offer. Additionally, you can easily print out your query using a UNIX command such as

```
a2ps q1.sql | lpr
```

b. It is also possible to redirect the output from queries to a file. For this purpose, issue the command

```
\o q1.output
```

from within `psql`, and run your query again. You will not see any output on the screen but if you open the file `q1.output` with the editor you will see the data there. Again, this is very helpful if you want to print the output.

If you issue further queries, then their output will be appended at the end of `q1.output`. This re-direction will stay in force until you log out of `psql` or you run the command

```
\o
```

without any argument. Output will now again appear directly on the screen.

c. You can save yourself some typing work by putting the following entries in your `.login` file:

```
setenv PGHOST mod-fund-databases
setenv PGDATABASE fundamentals
```

You can also create a `.psqlrc` file in your home directory and put in it any `psql` commands you want executed every time you start up. It is a good idea to put there:

```
\pset pager less
\pset fieldsep ','
```

The first line again tells PostgreSQL to use `less` as the pager, and the second line asks it to use commas for separating fields.

Question 3: More SQL queries

For ease of reference, here is a description of the various tables in "`fundamentals`" database:

```
  Table "staff"
 Attribute |         Type
-----------+----------------------
 sid       | integer
 title     | character varying(6)
 firstname | character varying(15)
 lastname  | character varying(20)
 email     | character varying(40)
 office    | integer
 phone     | integer
```

```
  Table "courses"
 Attribute |       Type
-----------+------------------
 level     | integer
 cid       | integer
 name      | character varying
 credits   | integer
 semester  | integer
 bc        | character(8)
```

```
  Table "lecturing"
 Attribute |  Type
-----------+--------
 cid       | integer
 sid       | integer
 year      | integer
 numbers   | integer
```

```
  Table "period"
 Attribute   |  Type
-------------+--------
 semester    | integer
 description | character varying
```

```
  Tables "allmarks--"
 Attribute  |     Type
------------+-------------
 student    | character(6)
 mark       | integer
 bc         | character(8)
```

The "`staff`" table holds information about staff members. The attribute "`sid`" (short for "staff id") is a unique integer for each record in the table.

The "`course`" table holds information about courses, with "`cid`" being a unique ID for the courses. Each course is at a "`level`" (1, 2, 3, or 4), carries a certain number of "`credits`" and is taught in a "`semester`" (1, 2, 3 or 4). The "`period`" table describes what the four "`semester`" codes mean. The attribute "`bc`" (for "banner code") is the ID number assigned by the University for the course, e.g., 06-21923 for "Fundamentals: Databases".

The "`lecturing`" table has information about which staff member taught which course in a given year. It also records the number of students in the course in that instance.

The "`allmarks03`" table (and other similar tables) store the marks received by the students in each course in a particular year. The course is videntified here by its banner code.

a. Use "`ORDER BY`" on the "`staff`" table in such a way that the details of Jim Yandle are listed somewhere at the top.

b. Which courses at level 3 or 4 carry more than 10 credits but do not take place in semester 2? (You must have a look at the table "`period`" first in order to understand the meaning of the integer in the "`semester`" field.)

c. For which staff members are either phone number or office number missing in the database? (See paragraph 14 in the Lecture Notes.)

d. Which staff members have a phone number in the database but no office number?

   You can do this in at least two ways: (i) using a single SELECT query with complex WHERE clause, and (ii) Using EXCEPT or EXCEPT ALL to combine multiple SELECT queires. Try both the approaches and see if there is a difference.

e. Which courses (listed by their banner codes "`bc`") were taken by students in 2003, judged by the existence of students taking their exams?

f. Which courses (listed by their "`cid`") where taught in 2003?

   Pause Does it match the answer to the previous question? If not, how on earth are we going to find out where the mismatch is?

g. Which courses (listed by their identifier "`cid`") were new in 2001 (ie, were not taught in 2000 or 1999)?