# Prac Exercise 02

## The PostgreSQL Catalog

## Aims

This simple exercise aims to get you to:

- become familiar with the PostgreSQL catalog
- understand what data is available to the query evaluator and storage manager

Pre-requisite: you have installed your PostgreSQL server following P01.

## Background

PostgreSQL uses its catalog tables to maintain a large amount of information that is used by the various components of the DB engine. As well as defining the user-level meta-data (names, types, constraints), the catalog tables also include information to assist the storage manager (e.g., size of attribute values), information to assist the query optimiser (e.g. size of table in pages), and so on. Some tables are global — shared by all databases on a PostgreSQL server — while others contain values local to a particular database.

Some of the more important tables (and some of their parameters are given below). Details on the other tables, and complete details of the given tables, are available in the PostgreSQL documentation

```
pg_authid(rolname, rolsuper, rolinherit, rolcreaterole, rolcreatedb, rolcatupdate,
        rolcanlogin, rolreplication, rolconnlimit, rolpassword, rolvaliduntil)

pg_database(datname, datdba, encoding, datcollate, datctype, datistemplate,
        datallowconn, datconnlimit, datlastsysoid, datfrozenxid, datminmxid,
        dattablespace, datacl)

pg_namespace(nspname, nspowner, nspacl)

pg_class(relname, relnamespace, reltype, reloftype, relowner, relam,
        relfilenode, reltablespace, relpages, reltuples, relallvisible,
        reltoastrelid, reltoastidxid, relhasindex, relisshared, relpersistence,
        relkind, relnatts, relchecks, relhasoids, relhaspkey, relhasrules,
        relhastriggers, relhassubclass, relfrozenxid, relminmxid, relacl, reloptions)

pg_attribute(attrelid, attname, atttypid, attstattarget, attlen, attnum, attndims,
        attcacheoff, atttypmod, attbyval, attstorage, attalign, attnotnull,
        atthasdef, attisdropped, attislocal, attinhcount, attcollation, attacl,
        attoptions, attfdwoptions)

pg_type(typname, typnamespace, typowner, typlen, typbyval, typtype, typcategory,
        typispreferred, typisdefined, typdelim, typrelid, typelem, typarray,
        typinput, typoutput, typreceive, typsend, typmodin, typmodout, typanalyze,
        typalign, typstorage, typnotnull, typbasetype, typtypmod, typndims,
        typcollation, typdefaultbin, typdefault, typacl)
```

## Exercise

In the lectures, I mentioned a PL/pgSQL function `schema()` that could use the PostgreSQL catalog tables to produce a list of tables/attributes for the `public` schema, in a format similar to that shown above. In fact, the above format was actually produced by an extension to the `schema()` function, which wraps lines before they become too long and hard to read.

The first thing to do is to make a copy of the `schema()` function:

```
$ mkdir some/directory/for/prac/p02
$ cd some/directory/for/prac/p02
$ cp /web/cs9315/19T2/pracs/p02/schema.sql .
# don't forget the dot, which means "current directory"
```

Create the beer database from Prac P01 (if it's not still there), and then do the following:

```
$ psql beer
psql (11.3)
Type "help" for help.

beer=# \i schema.sql ... loads contents of the file schema.sql ...
CREATE FUNCTION
beer=# select * from schema(); ... invokes the schema() function ...
            schema
-----------------------------
 bars(name, addr, license)
 beers(name, manf)
 drinkers(name, addr, phone)
 frequents(drinker, bar)
 likes(drinker, beer)
 sells(bar, beer, price)
(6 rows)

beer=#
```

Read the code for the function and make sure you understand how it works. You will most likely need to look at the documentation on PL/pgSQL for this. Once you understand how it works, make the following changes:

- change the name of the function to `schema1`
- make it return a set of tuples, rather than a set of text values

  ```
  create type SchemaTuple as ("table" text, "attributes" text)
  ```

  the value of the `attributes` field should still be a comma-separated string
- for each attribute in the list of attributes, add a description of its data type
- where required (e.g. `varchar` types), indicate the size of the value
- change the internal type names (e.g. `int4`) into more user-friendly names (e.g. `integer`)

Your new `schema1` function should produce output something like the following:

```
beer=# select * from schema1();
   table   |                      attributes
-----------+-----------------------------------------------------
 bars      | name:barname, addr:varchar(20), license:integer
 beers     | name:barname, manf:varchar(20)
 drinkers  | name:drinkername, addr:varchar(30), phone:char(10)
 frequents | drinker:drinkername, bar:barname
 likes     | drinker:drinkername, beer:beername
 sells     | bar:barname, beer:beername, price:float
```

if tested on the `beer` database from Prac P01.

Hint: you'll need to look at the PostgreSQL manual, especially the chapters on PL/pgSQL and System Catalog.

[Example Solution]

**End of Prac**

Let me know via the forums, or come to a consultation if you have any problems with this exercise ... *jas*