COMP3311 21T3 Database Systems

Assignment 1
Queries and Functions on BeerDB

Last updated: **Sunday 10th October 10:30am**

Most recent changes are shown in red ... older changes are shown in brown.

[Assignment Spec] [Database Design] [Examples] [Testing] [Submitting] [Fixes+Updates]

Aims

This assignment aims to give you practice in

- reading and understanding a small relational schema (BeerDB)
- implementing SQL queries and views to satisfy requests for information
- implementing PLpgSQL functions to aid in satisfying requests for information

The goal is to build some useful data access operations on the BeerDB database, which contains a wealth of information about everyone's* favourite beverage, One aim of this assignment is to use SQL queries (packaged as views) to extract such information. Another is to build PlpgSQL functions that can support higher-level activities, such as might be needed in a Web interface.

* well, mine anyway ...

Summary

Submission: Login to Course Web Site > Assignments > Assignment 1 > [Submit] > upload `ass1.sql`
or,
on a CSE server, `give cs3311 ass1 ass1.sql`

Required Files: `ass1.sql` (contains both SQL views and PLpgSQL functions)

Deadline: 21:00 Friday 15 October

Marks: **15 marks** toward your total mark for this course

Late Penalty: 0.1 *marks* off the ceiling mark for each hour late

How to do this assignment:

- read this specification carefully and completely
- create a directory for this assignment
- copy the supplied files into this directory
- login to `d.cse` and run your PostgreSQL server** (or run a PostgreSQL servr installed on your home machine)
- load the database and start exploring
- complete the tasks below by editing `ass1.sql`
- submit `ass1.sql` via WebCMS or `give`

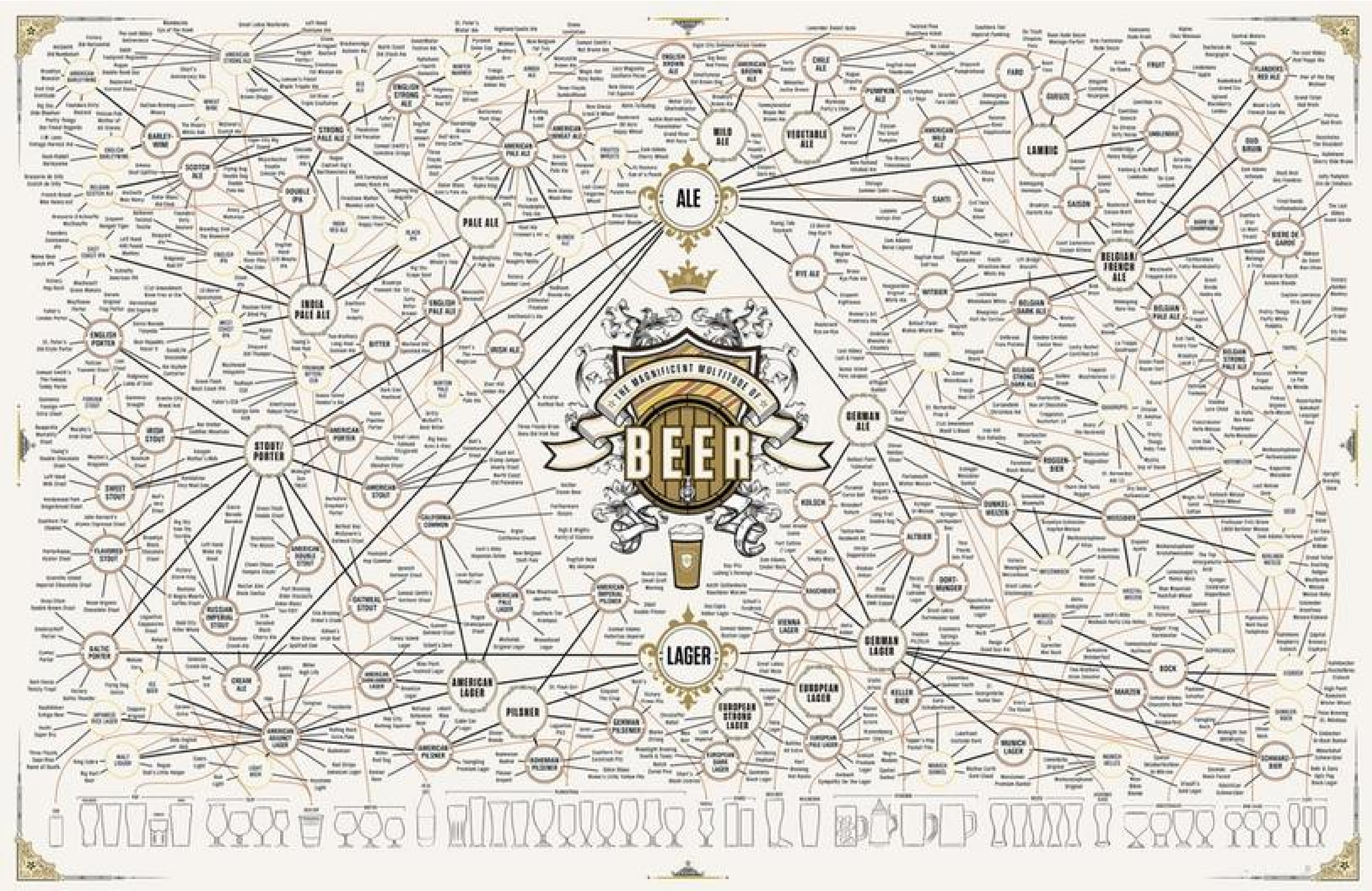
Details of the above steps are given below. Note that you can put the files wherever you like; they do not have to be under your `/srvr` directory. You also edit your SQL files on hosts other than `d.cse`. The only time that you need to use `d.cse` is to manipulate your database. Since you can work at home machine, you don't have to use `d.cse` at all while *developing* your solution, but you should definitely test it there before submitting.

Introduction

In order to work with a database, it is useful to have some background in the domain of data being stored. Here is a very quick tour of **beer**. If you want to know more, see the Wikipedia Beer Portal.

Beer is a fermented drink based on grain, yeast, hops and water. The grain is typically malted barley, but wide variety of other grains (e.g. oats, rye) can be used. There are a wide variety of beers, differing in the grains used, the yeast strain, and the hops. More highly roasted grains produce darker beers, different types of yeast produce different flavour profiles, and hops provide aroma and bitterness. To add even more variety, adjuncts (e.g. sugar, chocolate, flowers, pine needles, to name but a few) can be added.

The following diagram gives a hint of the variety of beer styles:



To build a database on beer, we need to consider:

- beer styles (e.g. lager, IPA, stout, etc., etc.)
- ingredients (e.g. varieties of hops and grains, and adjuncts)
- breweries, the facilities where beers are brewed
- beers, specific recipes following a style, and made in a particular brewery

Specific properties that we want to consider:

- ABV = alcohol by volume, a measure of a beer's strength
- IBU = international bitterness units
- each beer style has a range of ABVs for beers in that style
- for each beer, we would like to store
  - its name (brewers like to use bizarre or pun-based names for their beers)
  - its style, actual ABV, actual IBU (optional), year it was brewed
  - type and size of containers it's sold in (e.g. 375mL can)
  - its ingredients (usually a partial list because brewers don't want to reveal too much)
- for each brewery, we would like to store
  - its name, its location, the year it was founded, its website

The schema is described in more detail both as an ER model and an SQL schema in the schema page.

Doing this Assignment

The following sections describe how to carry out this assignment. Some of the instructions must be followed exactly; others require you to exercise some discretion. The instructions are targetted at people doing the assignment on `d.cse`. If you plan to work on this assignment at home on your own computer, you'll need to adapt the instructions to "local conditions".

If you're doing your assignment on the CSE machines, some commands must be carried out on `d.cse`, while others can (and probably should) be done on a CSE machine other than `d.cse`. In the examples below, we'll use `vxdb$` to indicate that the comand must be done on `d.cse` and `cse$` to indicate that it can be done elsewhere.

Setting Up

The first step in setting up this assignment is to set up a directory to hold your files for this assignment.

```
cse$ mkdir /my/dir/for/ass1
cse$ cd /my/dir/for/ass1
cse$ cp /home/cs3311/web/21T3/assignments/ass1/ass1.sql ass1.sql
cse$ cp /home/cs3311/web/21T3/assignments/ass1/ass1.dump ass1.dump
```

This gives you a template for the SQL views and PLpgSQL functions that you need to submit. You edit this file, (re)load the definitions into the database you created for the assignment, and test it there.

NOTE: ass1.sql is currently not complete

Speaking of the database, we have a modest sized database of all the beers that I've tasted over the last few years. We make this available as a PostgreSQL dump file. If you're working at home, you will need to copy it onto your home machine to load the database.

The next step is to set up your database:

```
... login to d.cse, source env, run your server as usual ...
... if you already had such a database
vxdb$ dropdb ass1
... create a new empty atabase
vxdb$ createdb ass1
```

```
... load the database, saving the output in a file called log
vxdb$ psql ass1 -f ass1.dump > log 2>&1
... check for error messages in the log; should be none
vxdb$ grep ERR log
... examine the database contents ...
vxdb$ psql ass1
```

The database loading should take less than 10 seconds on `d.cse`. The `ass1.dump` files contains the schema and data in a single file, along with a simple PLpgSQL function (`dbpop*()`.

If you're running PostgreSQL at home, you'll need to load both `ass1.sql` and `ass1.dump`.

You can grab the `check1.sql` script separately, once it becomes available.

Think of some questions you could ask on the database (e.g. like the ones in the Online Problem-solving Sessions) and work out SQL queries to answer them.

One useful query is

```
ass1=# select * from dbpop();
```

This will give you a list of tables and the number of tuples in each. The `dbpop()` function is written in PLpgSQL, and makes use of the PostgreSQL catalog. We'll look at this later in the term.

Your Tasks

Answer each of the following questions by typing SQL or PLpgSQL into the `ass1.sql` file. You may find it convenient to work on each question in a temporary file, so that you don't have to keep loading *all* of the other views and functions each time you change the one you're working on. Note that you can add as many auxuliary views and functions to

`ass1.sql` as you want. However, make sure that *everything* that's required to make all of your views and functions work is in the `ass1.sql` file before you submit.

Note #1: many of the queries are phrased in the singular e.g. "Find the beer that ...". Despite the use of "beer" (singular), it is possible that multiple beers satisfy the query. Because of this you should, in general, avoid the use of `LIMIT 1`.

Note #2: the database is not a complete picture of beers in the Real World. Treat each question as being prefaced by "According to the BeerDB database ...".

Note #3: you can assume that the names for styles and breweries are unique; you *cannot* assume this for beer names.

Q1 (2 marks)

What is the world's oldest brewery? Define an SQL view `Q1(brewery)` that gives its name.

Q2 (3 marks)

Nowadays, brewers often work together to make a beer. Such a beer is called a "collaboration beer" (or just "collab" for short) and is registered as being brewed by ~~both~~ all brewers. Define an SQL view `Q2(beer)` that gives the names of all collaboration beers.

Q3 (2 marks)

What is the worst beer in the world (determined by its rating)? Define a view `Q3(worst)` that gives its name. There may be several equally worst beers.

Q4 (2 marks)

Beers are brewed according to a style, which indicates what colour the beer should be, how strong it should be, etc. Occasionally brewers stray outside the bounds for a style e.g. make a beer stronger than the maximum ABV for that style. Define a view `Q4(beer,abv,style,max_abv)` that gives information about any beers whose ABV is higher than the maximum ABV for their style. The view should give the beer name, its ABV, its style, and the maximum ABV for that style.

Q5 (2 marks)

What *style* of beer is most commonly brewed (as determined by the number of beers brewed to that style)? Define a view `Q5(style)` that gives the name of the most common style.

Q6 (3 marks)

Sometimes data entry can go wrong and two slightly different versions of a style name can be entered into the database. The difference might be a spelling mistake or a mismatch in upper/lower-case letters in the name. Spelling mistakes are difficult to determine, but case mismatches are easy to detect. Define a view `Q6(style1,style2)` that determines pairs of style names that differ only in the upper/lower case of their letters. The order of style names matters in the result tuple; the lexicographically smaller style name should be in `style1`.

Q7 (2 marks)

The partial participation line between Brewery and BrewedBy in the ER model for this database suggests that there may be breweries that haven't (so far) brewed any beers. Define a view `Q7(brewery)` that finds any such breweries.

Q8 (3 marks)

Some cities (`metro` attribute) are known as "hot-spots" for breweries. Define a view `Q8(city,country)` that finds the city which has the most breweries located in it and the country where that city is located..

Q9 (3 marks)

Some breweries concentrate on a small number of beer varieties, others are prolific experimenters and make many different styles. Write a view `Q9(brewery,nstyles)` that gives the name and count of styles made by that brewery, for all breweries that make more than 5 different styles.

Q10 (3 marks)

Write a PLpgSQL function that takes a style name, and prints a list of all the beers that are brewed in that style.

```
create or replace function Q10(_style text) returns setof BeerInfo ...
```

where `BeerInfo` is a tuple type that could be defined as follows:

```
create type BeerInfo as (beer text, brewery text, style text, year YearValue, abv ABVvalue)
```

You could define the type like this but (HINT) you might find it convenient to have a `BeerInfo` view that returns tuples of this type, in which case the type would be defined automatically.

Note that the beer is a collaboration beer, all breweries should appear in the `brewery` attribute, separated by `' + '` and in alphabetical order on brewery name. Even if you include all of the collaborating breweries in the string, but they appear in a different order, your answer is not correct.

If the `_style` string given as an argument doesn't exactly match any beer style name, then you should simply return zero tuples.

Note that the underscore in the parameter name is important (i.e. should be `_style` not `style`). The reason for this will become obvious once you start writing the PLpgSQL code for the function.

There are examples of how the function should behave in the Examples page.

Q11 (4 marks)

Write a PLpgSQL function that takes a string as argument and finds all beers that contain that string in their name

```
create or replace function Q11(partial_name text) returns setof text ...
```

The function returns one string (`text`) for each beer whose name contains `partial_name` which is formatted as follows:

```
"Name of beer", Name of brewery or breweries, Beer style, abv value% ABV
```

You must use exactly the same punctuation and spacing as the above.

The breweries for collaboration beers must be handled the same as in Q10.

String match should be case-insensitive, e.g. `'al'` would match `'Pale Ale'`, `'Imperial Stout'` and many others.

There are examples of how the function should behave in the Examples page.

Q12 (6 marks)

Write a PostgreSQL function that takes a string as argument and gets information about all *breweries* that contain that string in their name:

```
create or replace function Q12(partial_name text) returns setof text ...
```

Each brewery has the following strings

```
Brewery name, founded Foundation-year
located in Location info
  "Name of beer", Beer style, Year brewed, abv_value% ABV
```

The third line is repeated once for each beer the brewery makes. If the brewery makes (so far) no beers, write a single line containing

```
  No known beers
```

You must use the same punctutation and spacing as above.

The "*Location info*" is a comma-separated string, determined as follows:

- the country is always the last element in the string
- if a region is known, include that in the location string
- if both town and metro are known, include just the town
- if only the metro is known, include that
- if only the town is known, include that

Examples:

```
Thornbury, Victoria, Australia        -- all four components are known
Marche-en-Famenne, Belgium            -- no region or metro
San Diego, California, United States  -- no town
Orange, New South Wales, Australia    -- no metro
```

String matching should be case-insensitive e.g. `'mo'` would match `'Chur (Behemoth) Brewing'`, `'Modern Times Beer'` and many others.

The breweries should appear in alphabetical order on their name. The list of beers should be arranged in ascending order of year. If there are multiple beers from a single year, order them by the beer name. If a beer is a collaboration beer, there is no need to mention the other brewery.

There are examples of how the function should behave in the Examples page.

Submission and Testing

We will test your submission as follows:

- create a testing subdirectory
- create a new database *TestingDB* and initialise it with `ass1.dump`
- load your work via the command: `psql TestingDB -f ass1.sql` (using your `ass1.sql`)
- run auto-marking on your views and functions using a script that we will eventually make available to you

Note that there is a time-limit on the execution of queries. If any query takes longer than 3 seconds to run (you can check this using the **\timing** flag) your mark for that query will be reduced.

Your submitted code must be *complete* so that when we do the above, your `ass1.sql` will load without errors. If your code does not work when installed for testing as described above and the reason for the failure is that your `ass1.sql` did not contain all of the required definitions, you will be penalised by a 1 mark administrative penalty.

Before you submit, it would be useful to test out whether the files you submit will work by following a similar sequence of steps to those noted above.

Have fun, *jas*