Assignment 1
Queries and Functions on BeerDB

Last updated: **Tuesday 14th March 9:50pm**
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:	23:59 Friday 17 March
Marks:	**12 marks** toward your total mark for this course
Late Penalty:	5% per day off the ceiling mark (deducted hourly) no submissions are permitted after 5 days

Assignment Project Exam Help
https://tutorcs.com
WeChat: cstutorcs

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 `vxdb2` and run your PostgreSQL server**
  (or run a PostgreSQL server installed on your home machine)
- load the database and start exploring
- complete the tasks below by editing `ass1.sql`
- test your work on `vxdb2`, which is where it's tested
- 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 `/localstorage` directory. You also edit your SQL files on hosts other than `vxdb2`. The only time that you need to use `vxdb2` is to manipulate your database. Since you can work at home machine, you don't have to use `vxdb2` 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:

ALE
LAGER
THE MAGNIFICENT MULTITUDE OF BEER

Assignment Project Exam Help
https://tutorcs.com
WeChat: cstutorcs

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 `vxdb2`. 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 `vxdb2`, while others can (and probably should) be done on a CSE machine other than `vxdb2`. In the examples below, we'll use `vxdb2$` to indicate that the comand must be done on `vxdb2` and `cse$` 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/23T1/assignments/ass1/ass1.sql ass1.sql
cse$ cp /home/cs3311/web/23T1/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.

Assignment Project Exam Help

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.

https://tutorcs.com

The next step is to set up your database:

WeChat: cstutorcs

```
... login to vxdb2, source env, run your server as usual ...
... if you already had such a database
vxdb2$ dropdb ass1
... create a new empty atabase
vxdb2$ createdb ass1
... load the database, saving the output in a file called log
vxdb2$ psql ass1 -f ass1.dump > log 2>&1
... check for error messages in the log; should be none
vxdb2$ grep ERR log
... examine the database contents ...
vxdb2$ psql ass1
```

The database loading should take less than 5 seconds on `vxdb2`. 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`.

Think of some questions you could ask on the database (e.g. like the ones in the lectures) 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.

There are examples of the results of each view and function in the Examples page.

Q0 (Style) (2 marks)

Given that you've already taken multiple programming courses, we should be able to assume that you'll express your code with good style conventions. But, just in case ...

You must ensure that your SQL queries follow a consistent style. The one I've been using in the lectures is fine. An alternative, where the word `JOIN` comes at the start of the line, is also OK. The main thing is to choose one style and use it consistently.

Assignment Project Exam Help

Similarly, PLpgSQL should be laid out like you would lay out any procedural programming language. E.g. bodies of loops should be indented from the `FOR` or `WHILE` statement that introduces them. E.g. the body of a function should be indented from the `BEGIN`...`END`.

https://tutorcs.com

Ugly, inconsistent layout of SQL queries and PLpgSQL functions will be penalised.

WeChat: cstutorcs

Q1 (2 marks)

Top rating beers are so delicious that you can't stop drinking once you open a container (bottle/can) containing one of them. Of course, you need to be careful how much alcohol you're consuming as you finish the contents of the container.

Define an SQL view `Q1(beer,"sold in",alcohol)` that gives the name of a highly rated beer (rating must be greater than 9), the size/kind of container that it's sold in, and what volume of alcohol is in that container. This can be computed by (volume * ABV / 100). Examples of the precise format of each tuple is shown on the examples page.

Q2 (2 marks)

Beer styles are defined by an international panel in terms of the ABV, the bitterness, the colour, and the aroma/flavours that you can expect in beers of that style. This database defines styles just in terms of minimum allowed ABV and maximum allowed ABV.

Define an SQL view `Q2(beer,style,abv,reason)` that gives a list of beers that are out-of-style (i.e. their ABV is either to low or too high for that style). For each such beer, give its name, the style that it claims to be, its actual ABV, and a reason why it is out-of-style. See the examples file for the format of the reasons. Use `numeric(4,1)` to define the format of ~~ABV values and~~ ABV differences.

Q3 (2 marks)

Define a view `Q3(country,"#beers")` that gives a list of all countries and the number of beers brewed in that country (according to this database). The list should include *all* countries; if a country makes no beers, give a count of zero. Assume that collaboration beers are brewed in both breweries, and count the countries for both breweries. Even if both breweries are in the same country, count it as two separate beers for that country.

Q4 (2 marks)

Define a view `Q4(beer,brewery,country)` that gives information about the worst beers in the world. "Worst" is determined by having a rating less than 3. Show the name of the offending beer, the offending brewery, and the country where the beer was made.

Q5 (2 marks)

Define a view `Q5(beer,ingredient,type)` that gives the name of beers that use ingredients whose origin in the Czech Republic.

Q6 (4 marks)

Define a view `Q6(beer)` that gives the names of beers that use both the most popular hop *and* most popular grain in their ingredients.

For the purposes of this exercise, treat "rolled" and "flaked" versions of grains as *different* from the base grain (e.g. "flaked wheat" is not the same as "wheat"). Similarly, treat "cryo" versions of hops and different from the base hop (e.g. "simcoe cryo" is not the same as "simcoe")

Q7 (2 marks)

Define a view `Q7(brewery)` that gives the names of any breweries that make no beers (according to this database).

Assignment Project Exam Help

Q8 (4 marks)

Write a PLpgSQL function that takes a beer id and returns the "full name" of the beer. The "full name" is formed by prepending (part of) the brewery name to the beer name.

https://tutorcs.com

```
create or replace function Q8(beer_id integer) returns text ...
```

WeChat: cstutorcs

You can work out beer IDs by looking the the `Beers` table. If you give an invalid beer ID, the function should return the string 'No such beer'.

It doesn't make sense to include `'Brewing Co'` or `'Beer Co'` or `'Brewery'` from the brewery name in the "full name", so you should filter these out using the regular expression `' (Beer|Brew).*$'`. If this filtering produces an empty string, use the complete name of the brewery.

An example: "Mountain Culture Beer Co" is a brewery that makes a beer called "Cult IPA". We want this to appear as "Mountain Culture Cult IPA" and *not* as "Mountain Culture Beer Co Cult IPA". Similarly, "Sierra Nevada Brewing Company" makes a beer called "Pale Ale". We want this to have the full name "Sierra Nevada Pale Ale".

If the beer is brewed collaboratively, all breweries should appear before the beer name in their shortened form, and separated by `' + '`. You can assume that no collaboration involves more than two breweries. The order that the breweries appear should be alphabetical on brewery name.

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

Q9 (6 marks)

Write a PostgreSQL function that takes a string as argument and gets information about all beers that contain that string somewhere in their name (use case-insensitive matching).

```
create or replace function Q9(partial_name text) returns setof BeerData ...
```

The `BeerData` type has three components:

- `beer`: the name of the beer
- `brewer`: the name of the brewery/breweries who make the beer
- `info`: the ingredients used in making the beer

Note that some beers involve two breweries who collaborate in making the beer. These beers should *not* be shown twice, once for each brewer. Instead, the `brewer` column should contain the names of all breweries in alphabetical order, and separated by `' + '`. There are examples of this in the Examples page.

The `info` should presented as a single text string, formatted as up to three lines: one containing a comma-separated list of hops, one containing a comma-separated list of grains, and one containing a comma-separated list of adjuncts. If no information is available about one of these types of ingredients, do not include a line for that type. Do not include a final `'\n'` character in the result string.

An example of what the `info` should look like for a beer that uses all ingredient types:

```
Hops: Bravo,Centennial,Mosaic
Grain: Oats,Pale,Rye,Treticale,Wheat
Extras: Lactose,Vanilla
```

The comma-separated ingredient lists should be in alphabetical order.

For collaboration beers, both breweries should appear, in alphabetical order and separated by `' + '`. For this question, you *cannot* assume that collaborations involve only two breweries.

Note that `psql` puts a `'+'` at the end of each line to indicate that the string spans multiple lines. Ignore this; it's an output artifact.

Assignment Project Exam Help

There are more examples of how the function should behave in the Examples page. In particular, if there are no beers matching the `partial_name`, simply return an empty table (0 rows).

https://tutorcs.com

Submission and Testing

WeChat: cstutorcs

Submit your completed `ass1.sql` using `give` or Webcms3. We will test your submission by first creating a new beer database, and then loading your code into it, and running your views and functions. We will conduct further tests by loading a slight different version of the database (same schema, different data), loading your code into that database and running further tests. More details of this are given in the submission and testing

pages.

Important note: if your code does not load correctly into a freshly created database, and we need to fix it to make it load, you will be penalised 2 marks.

Have fun, *jas*