|  |  |  |
| --- | --- | --- |
| [COMP9311 18s2](http://www.cse.unsw.edu.au/~cs9311) | **Lab 4** **SQL Queries, Views, and Aggregates (ii)** | [Database Systems](http://www.cse.unsw.edu.au/~cs9311) |

**Aims**

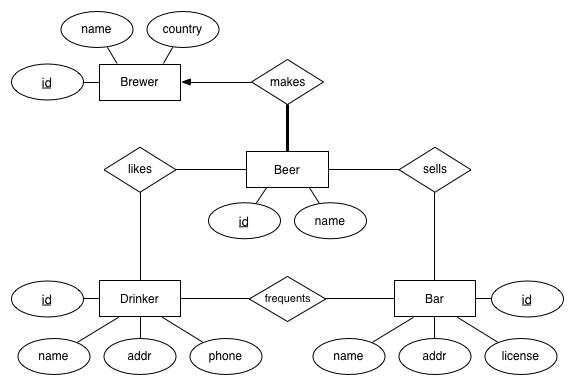
This exercise aims to give you practice in:

* asking SQL queries on a relatively simple schema
* using SQL aggregation and grouping operators
* writing SQL view definitions

This exercise will **not** explain how to do everything in fine detail. Part of the aim of the exercise is that you explore how to use the PostgreSQL system. The documentation for this system contains much useful information:[PostgreSQL Manual](https://www.postgresql.org/docs/current/static/index.html). You should become familiar with where to find useful information in the documentation; you will need to know how to use PostgreSQL for the assignments and the exam.

**Background**

In lectures, we used a simple database about beers, bars and drinkers to illustrate aspects of querying in SQL. The database was designed to simplify queries by using symbolic primary keys (e.g., Beers.name). In practice, we don't use symbolic primary keys because: (a) they typically occupy more space than a numeric key, (b) symbolic names have an annoying tendency to change over time (e.g., you might change your email, and having email as a primary key creates a multitude of update problems). Therefore, we have designed a (slightly) more realistic schema for representing the same information:



The SQL schema will obviously be different to the schema used in lectures, and is available in the file:

[/home/cs9311/web/18s2/labs/04/schema.sql](https://www.cse.unsw.edu.au/~cs9311/18s2/labs/04/schema.sql)

If you are working on your laptop (and not via putty), you can grab copies of all files used in this Prac in the ZIP archive:

[/home/cs9311/web/18s2/labs/04/prac.zip](https://www.cse.unsw.edu.au/~cs9311/18s2/labs/04/prac.zip)

**Setting up the PostgreSQL Database**

Login to a machine with a PostgreSQL server running. If you already have a beer2 database and you want to replace it, you will, of course, need to drop it first:

$ **dropdb beer2**

Then do the following:

$ **createdb beer2**

$ **psql beer2 -f schema.sql**

$ **psql beer2 -f data.sql**

You now have a database that you can use via:

$ **psql beer2**

**Exercises**

Use the database you created above to do the exercises below. The aim is to get practice in building queries in PostgreSQL.

In the questions below, you are required to produce SQL queries to solve a range of data retrieval problems on this schema. For each problem, create a view called Q*n* which holds the top-level SQL statement that produces the answer (this SQL statement may make use of any views defined earlier in the Lab). In producing a solution for each problem, you may define as many auxiliary views as you like.

To simplify the process of producing these views, a template file ([queries.sql](https://www.cse.unsw.edu.au/~cs9311/18s2/labs/04/queries.sql)) is available. While you're developing your views, you might find it convenient to edit the views in one window (i.e., edit the queries.sql file containing the views) and copy-and-paste the view definitions into another window running psql.

Note that the order of tuples in the results does not matter. As long as you have the same set of tuples, your view is correct. Remember that, in theory, the output from an SQL query is a set. Some test queries use an explicit ordering, but that should not be included in the view definition.

Note also that the sample outputs typically use column names that are different to the column names in the table. You should use the column names given in the sample output; treat them as part of description of the question.

Once you have completed each of the view definitions, you can test it simply by typing:

beer2=# **select \* from Q*n*;**

and observing whether the result matches the expected result given below.

**Queries on Beer Database v2**

Write an SQL view to answer each of the following queries. Note that *none* of your queries should contain internal id values; all references to entities in queries should be via their name.

1. ***What beers are made by Toohey's?***

In PostgreSQL, the results should look like:

beer2=# **select \* from Q1;**

beer

-------------

New

Old

Red

Sheaf Stout

(4 rows)

1. ***Show beers with headings "Beer", "Brewer".***

In PostgreSQL, the results should look like:

beer2=# **select \* from Q2;**

Beer | Brewer

--------------------------+---------------------

80/- | Caledonian

Amber Ale | James Squire

Bigfoot Barley Wine | Sierra Nevada

Burragorang Bock | George IV Inn

Chestnut Lager | Bridge Road Brewers

Crown Lager | Carlton

Fosters Lager | Carlton

India Pale Ale | James Squire

Invalid Stout | Carlton

Melbourne Bitter | Carlton

New | Toohey's

Nirvana Pale Ale | Murray's

Old | Toohey's

Old Admiral | Lord Nelson

Pale Ale | Sierra Nevada

Pilsener | James Squire

Porter | James Squire

Premium Lager | Cascade

Red | Toohey's

Sink the Bismarck | Brew Dog

Sheaf Stout | Toohey's

Sparkling Ale | Cooper's

Stout | Cooper's

Tactical Nuclear Penguin | Brew Dog

Three Sheets | Lord Nelson

Victoria Bitter | Carlton

(26 rows)

1. ***Find the brewers whose beers John likes.***

In PostgreSQL, the results should look like:

beer2=# **select \* from Q3;**

brewer

---------------

Brew Dog

James Squire

Lord Nelson

Sierra Nevada

Caledonian

(5 rows)

1. ***How many different beers are there?***

In PostgreSQL, the results should look like:

beer2=# **select \* from Q4;**

#beers

--------

26

(1 row)

1. ***How many different brewers are there?***

In PostgreSQL, the results should look like:

beer2=# **select \* from Q5;**

#brewers

----------

12

(1 row)

1. ***Find pairs of beers by the same manufacturer*** (but no (a,b) and (b,a) pairs, or (a,a))

In PostgreSQL, the results should look like:

beer2=# **select \* from Q6;**

beer1 | beer2

---------------------+--------------------------

Amber Ale | Porter

Amber Ale | Pilsener

Amber Ale | India Pale Ale

Bigfoot Barley Wine | Pale Ale

Crown Lager | Victoria Bitter

Crown Lager | Melbourne Bitter

Crown Lager | Invalid Stout

Crown Lager | Fosters Lager

Fosters Lager | Victoria Bitter

Fosters Lager | Melbourne Bitter

Fosters Lager | Invalid Stout

India Pale Ale | Porter

India Pale Ale | Pilsener

Invalid Stout | Victoria Bitter

Invalid Stout | Melbourne Bitter

Melbourne Bitter | Victoria Bitter

New | Sheaf Stout

New | Red

New | Old

Old | Sheaf Stout

Old | Red

Old Admiral | Three Sheets

Pilsener | Porter

Red | Sheaf Stout

Sink the Bismarck | Tactical Nuclear Penguin

Sparkling Ale | Stout

(26 rows)

1. ***How many beers does each brewer make?***

In PostgreSQL, the results should look like:

beer2=# **select \* from Q7 order by brewer;**

brewer | nbeers

---------------------+--------

Brew Dog | 2

Bridge Road Brewers | 1

Caledonian | 1

Carlton | 5

Cascade | 1

Cooper's | 2

George IV Inn | 1

James Squire | 4

Lord Nelson | 2

Murray's | 1

Sierra Nevada | 2

Toohey's | 4

(12 rows)

1. ***Which brewer makes the most beers?***

In PostgreSQL, the results should look like:

beer2=# **select \* from Q8;**

brewer

---------

Carlton

(1 row)

1. ***Beers that are the only one by their brewer.***

In PostgreSQL, the results should look like:

beer2=# **select \* from Q9;**

beer

------------------

80/-

Burragorang Bock

Chestnut Lager

Nirvana Pale Ale

Premium Lager

(5 rows)

1. ***Beers sold at bars where John drinks.***

In PostgreSQL, the results should look like:

beer2=# **select \* from Q10 order by beer;**

beer

-------------------

Burragorang Bock

New

Old

Old Admiral

Pale Ale

Sink the Bismarck

Sparkling Ale

Three Sheets

Victoria Bitter

(9 rows)

You might like to consider a variation on this query to find just the beers that John likes that are sold in the bars where he drinks. The solution is given in the solutions file.

1. ***Bars where either Gernot or John drink.***

In PostgreSQL, the results should look like:

beer2=# **select \* from Q11 order by bar;**

bar

------------------

Australia Hotel

Coogee Bay Hotel

Local Taphouse

Lord Nelson

Royal Hotel

(5 rows)

1. ***Bars where both Gernot and John drink.***

In PostgreSQL, the results should look like:

beer2=# **select \* from Q12;**

bar

-------------

Lord Nelson

(1 row)

1. ***Bars where John drinks but Gernot doesn't***

In PostgreSQL, the results should look like:

beer2=# **select \* from Q13;**

bar

------------------

Australia Hotel

Local Taphouse

Coogee Bay Hotel

(3 rows)

1. ***What is the most expensive beer?***

In PostgreSQL, the results should look like:

beer2=# **select \* from Q14;**

beer

-------------------

Sink the Bismarck

(1 row)

1. ***Find bars that serve New at the same price as the Coogee Bay Hotel charges for VB.***

In PostgreSQL, the results should look like:

beer2=# **select \* from Q15;**

bar

-------------

Royal Hotel

(1 row)

1. ***Find the average price of common beers*** (where "common" = served in more than two hotels)

In PostgreSQL, the results should look like:

beer2=# **select \* from Q16;**

beer | AvgPrice

-----------------+----------

Victoria Bitter | 2.40

New | 2.59

Old | 2.68

(3 rows)

1. ***Which bar sells 'New' cheapest?***

In PostgreSQL, the results should look like:

beer2=# **select \* from Q17;**

bar

--------------

Regent Hotel

(1 row)

1. ***Which bar is most popular?*** (Most drinkers)

In PostgreSQL, the results should look like:

beer2=# **select \* from Q18;**

bar

------------------

Coogee Bay Hotel

Lord Nelson

(2 rows)

1. ***Which bar is least popular?*** (May have no drinkers)

In PostgreSQL, the results should look like:

beer2=# **select \* from Q19;**

bar

-----------------

Local Taphouse

Marble Bar

Regent Hotel

Australia Hotel

Royal Hotel

(5 rows)

1. ***Which bar is most expensive?*** (Highest average price)

In PostgreSQL, the results should look like:

beer2=# **select \* from Q20;**

bar

----------------

Local Taphouse

(1 row)

1. ***Which beers are sold at all bars?***

In PostgreSQL, the results should look like:

beer2=# **select \* from Q21;**

beer

------

(0 rows)

i.e. no beers are sold at all bars.

1. ***Price of cheapest beer at each bar?***

In PostgreSQL, the results should look like:

beer2=# **select \* from Q22;**

bar | min\_price

------------------+-----------

Coogee Bay Hotel | 2.25

Local Taphouse | 7.50

Royal Hotel | 2.30

Australia Hotel | 3.00

Regent Hotel | 2.20

Marble Bar | 2.80

Lord Nelson | 3.00

(7 rows)

1. ***Name of cheapest beer at each bar?***

In PostgreSQL, the results should look like:

beer2=# **select \* from Q23;**

bar | beer

------------------+-----------------

Australia Hotel | New

Coogee Bay Hotel | New

Lord Nelson | New

Marble Bar | New

Marble Bar | Victoria Bitter

Regent Hotel | New

Regent Hotel | Victoria Bitter

Royal Hotel | Victoria Bitter

Royal Hotel | New

Local Taphouse | Pale Ale

(10 rows)

1. ***How many drinkers are in each suburb?***

In PostgreSQL, the results should look like:

beer2=# **select \* from Q24;**

addr | count

----------+-------

Randwick | 1

Mosman | 1

Newtown | 1

Clovelly | 1

(4 rows)

1. ***How many bars in suburbs where drinkers live?*** (must include suburbs with no bars)

In PostgreSQL, the results should look like:

beer2=# **select \* from Q25;**

addr | #bars

----------+-------

Randwick | 1

Mosman | 0

Newtown | 0

Clovelly | 0

(4 rows)

You should attempt the above exercises before looking at the sample solutions which will be released at the end of the week.