Lab: Week 15

36-350 – Statistical Computing

Week 15 – Fall 2020

Name: Kimberly Zhang

Andrew ID: kyz

You must submit **your own** lab as a knitted PDF file on Gradescope.

Question 1

(10 points)

Notes 12B (3) + Notes 14A (6)

Create a new table called `widgets` and populate it exactly as you did the `rdata` table in Lab 14, except rename columns `a` and `b` as `code` and `category`, respectively. Display the first five rows of your table.

```
postgres=# create table widgets (
id serial primary key,
code text,
category text,
moment date,
x real);
CREATE TABLE
postgres=# insert into widgets (code, category, moment, x)
(select md5(id::varchar(3)) as code,
('{X, Y, Z}'::text[])[(random()*2 + 1)::integer] as category,
'2020-01-01'::date + (random()*365)::integer as moment, random()*100 as x
from generate_series(1,100) as id);
INSERT 0 100
postgres=# select * from widgets limit 5;
 id |               code               | category |   moment   |    x
----+----------------------------------+----------+------------+----------
  1 | c4ca4238a0b923820dcc509a6f75849b | X        | 2020-01-24 | 69.25622
  2 | c81e728d9d4c2f636f067f89cc14862c | X        | 2020-12-04 | 67.49018
  3 | eccbc87e4b5ce2fe28308fd9f2a7baf3 | Y        | 2020-02-26 |  36.5877
  4 | a87ff679a2f3e71d9181a67b7542122c | X        | 2020-03-07 | 83.44778
  5 | e4da3b7fbbce2345d7772b0674a318d5 | Z        | 2020-10-13 | 43.10411
(5 rows)
```

Question 2

(10 points)

Notes 12B (3-4,6) + Notes 14A (6)

Create a new table `shipments` that should include `id` that acts as the serial primary key for the table, an `integer` *foreign key* `widget_id` that references `widgets`, a text `location`, and an `integer`-valued `number_shipped` that has default value zero. Populate the `widget_id` and `number_shipped` columns with random integers (between 1 to 100 for `widget_id` and between 0 and 50 for `number_shipped`), and the `location` column with cities randomly chosen from the list {Pittsburgh,New York,Vancouver,Austin}. There should be 50 rows in all. You should `update` the rows where `widget_id` is less than 5 so as to have a null `widget_id`; these rows then represent shipments that did not include any widgets at all. Display the first five rows of your table.

```
postgres=# create table shipments(id serial primary key,
widget_id integer references widgets (id), location text,
number_shipped integer default 0);
CREATE TABLE
postgres=# insert into shipments (widget_id, location, number_shipped)
(select (random()*99 + 1)::integer, ('{Pittsburgh,New York, Vancouver,Austin}'::text[])[(random()*3+1):
(random()*50)::integer
from generate_series(1, 50));
INSERT 0 50
postgres=# update shipments set widget_id = NULL where widget_id < 5;
UPDATE 2
postgres=# select * from shipments limit 5;
 id | widget_id | location  | number_shipped
----+-----------+-----------+----------------
 51 |        91 | Austin    |             18
 52 |        89 | New York  |              2
 53 |        60 | New York  |              2
 54 |        72 | Vancouver |             40
 55 |        67 | Vancouver |             27
(5 rows)
```

Question 3

(10 points)

Notes 14A (5) + Notes 15A (4) + Notes 15B (3-6)

Using a `join`, select the shipment id, widget id, widget code, widget moment, and shipment location for all the shipments of widgets of `category 'X'`. Sort the result in descending order by `moment`. Remember that if two tables have columns with the same name, you need to disambiguate: for instance, use `shipments.id` and `widgets.id` rather than just `id`. Otherwise, `postgres` will figure out which table the named column is in. (However, it doesn't hurt to use `<table name>.<column name>` even if you don't have to.)

```
postgres=# select s.id as "shipment id", w.id as "widget id", w.code as "widget code", w.moment as "wid
from shipments s join widgets w on s.widget_id = w.id
where w.category = 'X'
order by moment desc;
 shipment id | widget id |           widget code            | widget moment | shipment location
-------------+-----------+----------------------------------+---------------+-------------------
          73 |        88 | 2a38a4a9316c49e5a833517c45d31070 | 2020-12-06    | New York
          94 |         8 | c9f0f895fb98ab9159f51fd0297e236d | 2020-10-25    | Vancouver
          65 |        39 | d67d8ab4f4c10bf22aa353e27879133c | 2020-09-14    | New York
          91 |        36 | 19ca14e7ea6328a42e0eb13d585e4c22 | 2020-07-02    | New York
          71 |        21 | 3c59dc048e8850243be8079a5c74d079 | 2020-06-28    | Austin
          83 |        21 | 3c59dc048e8850243be8079a5c74d079 | 2020-06-28    | Vancouver
          87 |        80 | f033ab37c30201f73f142449d037028d | 2020-06-13    | New York
          75 |         6 | 1679091c5a880faf6fb5e6087eb1b2dc | 2020-05-15    | Pittsburgh
          85 |        33 | 182be0c5cdcd5072bb1864cdee4d3d6e | 2020-05-02    | Vancouver
          77 |        67 | 735b90b4568125ed6c3f678819b6e058 | 2020-04-17    | Vancouver
```

```
         55 |        67 | 735b90b4568125ed6c3f678819b6e058 | 2020-04-17    | Vancouver
         99 |        81 | 43ec517d68b6edd3015b3edc9a11367b | 2020-03-11    | Austin
         56 |        81 | 43ec517d68b6edd3015b3edc9a11367b | 2020-03-11    | New York
         98 |        91 | 54229abfcfa5649e7003b83dd4755294 | 2020-02-01    | New York
         51 |        91 | 54229abfcfa5649e7003b83dd4755294 | 2020-02-01    | Austin
(15 rows)
```

Question 4

(10 points)

Notes 15A (3-4)

Using `group by`, print the minimum and maximum `moment` for the widgets in each category, ordered by the minimum value of `moment`.

```
postgres=# select category, min(moment) as "min moment",
max(moment) as "max moment"
from widgets
group by category
order by min(moment);
 category | min moment | max moment
----------+------------+------------
 Z        | 2020-01-02 | 2020-10-13
 Y        | 2020-01-10 | 2020-12-28
 X        | 2020-01-24 | 2020-12-25
(3 rows)
```

Question 5

(10 points)

Notes 14A (5) + Notes 15A (3-4) + Notes 15B (4)

Using a `join`, a `where` clause, a `group by` clause, and the `count()` and `avg()` functions, print the category, the number of shipments with widgets in each category along with the average value of `x`, ordered by the average value of `x`. (Using `as`, rename the widget count in your output to be `shipment_count` and the average value of `x` to be `avg_x`.) If you are confused about the `where` clause: you don't want to consider rows in `shipments` where the `widget_id` is NULL.

```
postgres=# select w.category as category,
count(s.id) as "shipment_count", avg(w.x) as "avg_x"
from widgets w join shipments s on w.id = s.widget_id
where s.widget_id is not NULL
group by w.category
order by avg(w.x);
 category | shipment_count |       avg_x
----------+----------------+--------------------
 Y        |             26 | 48.405415672522324
 Z        |              9 |  48.52800316280789
 X        |             15 | 56.541674550374346
(3 rows)
```

Question 6

(10 points)

Notes 14A (5) + Notes 15A (3-4) + Notes 15B (4)

Using a `join`, a `where` clause, a `group by` clause, and the `count()` function, print the category, location, and number of shipments with widgets for each combination of category and location, ordered first by category and then by location. As above, rename the count to be `shipment_count`.

```
postgres=# select w.category as category, s.location as location, count(s.id) as "shipment_count"
from widgets w join shipments s on w.id = s.widget_id
where s.widget_id is not NULL
group by w.category, s.location
order by w.category, s.location;
 category |  location  | shipment_count
----------+------------+----------------
 X        | Austin     |              3
 X        | New York   |              6
 X        | Pittsburgh |              1
 X        | Vancouver  |              5
 Y        | Austin     |              4
 Y        | New York   |              8
 Y        | Pittsburgh |              4
 Y        | Vancouver  |             10
 Z        | Austin     |              2
 Z        | New York   |              4
 Z        | Pittsburgh |              3
(11 rows)
```

Question 7

(10 points)

Notes 14A (4-5) + Notes 15A (3-4) + Notes 15B (4)

Using a `join`, a `where` clause, a `group by` clause, and the `count()` and `avg()` functions, print the category and the number of shipments with widgets in each category, only counting those shipments that contained at least the average number of widgets shipped overall, ordered by the average number shipped. As above, rename the count as `shipment_count`. (Hint: to calculate and use the average of `number_shipped` in the `where` clause, remember that you use `select` to do on the fly calculations. Just surround your `select` clause with parentheses, so as to not confuse the parser.)

```
postgres=# select w.category as category, count(s.id) as "shipment_count"
from widgets w join shipments s on w.id = s.widget_id
where s.widget_id is not NULL
and s.number_shipped >= (select avg(number_shipped) from shipments)
group by w.category
order by avg(s.number_shipped);
 category | shipment_count
----------+----------------
 Z        |              4
 X        |             10
 Y        |             13
(3 rows)
```

Question 8

(10 points)

Notes 15A (3-5) + Notes 15B (4)

Using a `join`, `group by` and `having` clauses, and the `count()` and `avg()` functions, print the category, average number shipped (rounded to one digit), and the number of shipments with widgets in each category, for categories where the average value of `number_shipped` is bigger than some threshold you choose (e.g., 25), in *descending* order of average `number_shipped`. (There is no need for a `where` clause here: you are filtering *after* you group, not before.) Rename the rounded average of `number_shipped` as `avg_shipped` and the count as `shipment_count`.

```
postgres=# select w.category as category, round(avg(s.number_shipped),1) as "avg_shipped", count(s.id) a
from widgets w join shipments s on w.id = s.widget_id
group by w.category
having avg(s.number_shipped) > 25
order by avg(s.number_shipped) desc;
 category | avg_shipped | shipment_count
----------+-------------+----------------
 X        |        27.1 |             15
(1 row)
```

Cut and paste the following into your `postgres` session.

```
create table continents (
  code char(3) primary key,
  continent text
);

insert into continents values
  ('usa','North America'),
  ('can','North America'),
  ('gbr','Europe'),
  ('ger','Europe'),
  ('chn','Asia'),
  ('ind','Asia'),
  ('egy','Africa'),
  ('ecu','South America');

create table capitals (
  code char(3) primary key,
  capital text
);

insert into capitals values
  ('mex','Mexico City'),
  ('can','Ottawa'),
  ('jpn','Tokyo'),
  ('ind','New Delhi'),
  ('ecu','Quito');
```

Question 9

(10 points)

Notes 15B (3-6)

Display the values of `capital` and `continent` such that every value of `continent` in the `continents` table is displayed, and such that there are no `null` values in the output `continent` column. There thus will be eight rows in all. (Note that the type of `join` you use will be dictated on the order that you mention the tables in the `select` command below, i.e., there is nothing inherently `left` or `right` about either table.)

```
postgres=# select continent, capital
from capitals as ca right join continents co on ca.code = co.code;
   continent   |  capital
---------------+-----------
 North America |
 North America | Ottawa
 Europe        |
 Europe        |
 Asia          |
 Asia          | New Delhi
 Africa        |
 South America | Quito
(8 rows)
```

Question 10

(10 points)

Notes 15B (3-6)

Display only the capital city names for countries that do not appear in the `continents` table. (Note that this is a “set difference,” but `postgres` doesn’t have an explicit construction for determining one, other than by using an `except` construction. Here, using a `where` that determines whether particular values are `null` or not will achieve the same goal.)

```
postgres=# select capital
from capitals as ca left join continents co on ca.code = co.code
where co.continent is null;
   capital
-------------
 Mexico City
 Tokyo
(2 rows)
```