More of Grep

We have file.txt as following

```
Thus
This is the fourth line
127.45.199.001  network*subnetwork
hohoho aaaaa
601-271-6649   834-958-2261
cat hat that  mat
bat chat      zat
josephSmith2019@data.210  peterJobs2@data210.edu peter2Jobs@data.edu
```

We can also use the square bracket notation, which would allow the regular expression to use any of the letters inside the square brackets. For example, to find matches for any of these three words: 'cat', 'bat', and 'hat'; then we can write the following command:

```
[/data/L05]$ grep --color -E '[cbhd]at' file.txt
cat hat that  mat
bat chat      zat
```

Question: how to write grep command that find only 'chat' and 'that'

We can also specify the number/range of consecutive occurrences for whatever is inside the square brackets.

```
[/data/L05]$ grep --color -E '[cbht]{2}at' file.txt
cat hat that  mat
 bat chat      zat
```

The command above will match words like 'that', 'chat', …

```
[/data/L05]$ grep --color -E '[cbht]{,2}at' file.txt
cat hat that  mat
bat chat      zat
josephSmith2019@data.210  peterJobs2@data210.edu peter2Jobs@data.edu
```

We can also specify ranges inside the square brackets; for example, to match any letter between 'a' and 'p', then we can do the following:

```
[/data/L05]$ grep --color -E '[a-p]{2}at' file.txt
bat chat      zat
```

In addition to that, we can specify multiple ranges or multiple single characters or a combination of both inside the square brackets.

The command below for example, will match any letter between 'a' and 'p', or the letter 'z', or any number between 0 and 9

```
[/data/L05]$ $ grep --color -E '[a-pz0-9]{2}at' file.txt
bat chat      zat
```

So, to match a phone number that's in this format xxx-xxx-xxxx, we can write the following command:

```
    grep --color -E '[7-9][0-9]{2}-[0-9]{3}-[0-9]{4}' file.txt
or  grep --color -E '[7-9][0-9][0-9]-[0-9][0-9][0-9]-[0-9][0-9][0-9][0-9]' file.txt
```

```
[/data/L05]$ grep --color -E '[7-9][0-9]{2}-[0-9]{3}-[0-9]{4}' file.txt
601-271-6649  834-958-2261
```

```
Or grep --color -E '8[0-9]{2}-[0-9]{3}-[0-9]{4}' file.txt
```

If we have file f2.txt as following:

```
27.18 2.0 Female Yes Sat Lunch 2ch 27.123.ch
```

```
[/data/L05]$ grep -oE '\w{3}' f2.txt | tr '\n' '-'
Fem-ale-Yes-Sat-Lun-2ch-123-
```

Any special characters like '.', '*', '@' etc. will be treated as delimilator like space ' ' when using grep

```
[/data/L05]$ grep -oE '\w{3}' file.txt | tr '\n' '-'
Thu-Thi-the-fou-rth-lin-127-199-001-net-wor-sub-net-wor-hoh-oho-aaa-601-
```

271-664-834-958-226-cat-hat-tha-mat-bat-cha-zat-jos-eph-Smi-th2-019-dat-210-pet-erJ-obs-dat-a21-edu-pet-er2-Job-dat-edu-

Replacing and delete values tr

To replace space with '_', we can do the following:

```
[/data/L05]$ echo 'hello world!' | tr ' ' '_'
hello_world!
```

To delete space, we can do the following:

```
[/data/L05]$ echo 'hello world!' | tr -d ' '
helloworld!
```

To delete a character (say 'i') using 'tr', we can do the following:

```
[/data/L05]$ $ tr -d 'i' < hm.txt
Ths s the frst lne
LetMeIgnoreSome spaces here
89Hello There...
bang bang da
```

To delete the characters 'i', 'a', and 'o', we can do the following:

```
[/data/L05]$ tr -d '[iao]' < hm.txt
Ths s the frst lne
LetMeIgnreSme spces here
89Hell There...
bng bng d
```

To delete all characters except 'i', 'a', or 'o', we use -c (complement) option:

```
 [/data/L05]$ tr -d -c '[iao]' < hm.txt
 iiiiooaoaaa
```

To delete all punctuation , we use -c option:

```
[/data/L05]$ echo 'Hello,  world!' | tr -d -c '[a-zA-Z]'
Helloworld
```

To delete all punctuation except space, we use -c option:

```
[/data/L05]$ echo 'Hello,  world!' | tr -d -c '[a-zA-Z ]'
Hello world
```

```
[/data/L05]$ seq -w 8 11

08
09
10
11
```

We can also print the sequences as part of another string, by using 'seq' with the option -f.
Example:

```
[/data/L05]$ seq -f "This is line number: %g" 8 11
This is line number: 8
This is line number: 9
This is line number: 10
 This is line number: 11
```

```
[/data/L05]$ seq -f "Line %g" 10 > lines.txt
[/data/L05]$ cat lines.txt
Line 1
Line 2
Line 3
Line 4
Line 5
Line 6
Line 7
Line 8
Line 9
Line 10
```

March 09, 2020 DATA 210: Notes

Using 'sed' to filter lines

Check http://www.grymoire.com/Unix/Sed.html#uh-33 for sed tutorial

Here is a list of the cases/examples we covered in class:

The command below will print all the lines; however, it will print the lines from 1 to 3 twice. (p duplicate printing line)

```
[/data/L05]$ cat -n lines.txt | sed '1,3p'
```

```
or [/data/L05]$  sed '1,3p' lines.txt
```

The command below will print the lines from 1 to 3

```
[/data/L05]$ cat -n lines.txt | sed -n '1,3p'
```

```
 or [/data/L05]$ sed -n '1,3p' lines.txt
```

```
 or [/data/L05]$ head -n 3 lines.txt
```

```
 or [/data/L05]$ < lines.txt awk 'NR<=3'
```

The "-n" option will not print anything unless an explicit request to print is found

The command below will print the lines starting at

```
line 4
```

```
[/data/L05]$ < lines.txt  sed '1,3d'
```

The “d” deletes every line that matches the restriction

The command below will also print the lines starting

at line 4

[/data/L05]$ < lines.txt sed -n '1,3!p'

Or [/data/L05]$ < lines.txt awk 'NR>=4'

Or [/data/L05]$ < lines.txt tail -n +4

Using tail -n +4 to remove the first 3 lines

and tail is much faster

Using 'sed' to filter lines

Check http://www.grymoire.com/Unix/Sed.html#uh-33 for sed tutorial

Here is a list of the cases/examples we covered in class:

The command below will print all the lines; however, it will print the lines from 1 to 3 twice. (p duplicate printing line)

```
[/data/L05]$ cat -n lines.txt | sed '1,3p'
```

or `[/data/L05]$ sed '1,3p' lines.txt`

The command below will print the lines from 1 to 3

```
[/data/L05]$ cat -n lines.txt | sed -n '1,3p'
```

or `[/data/L05]$ sed -n '1,3p' lines.txt`

or `[/data/L05]$ head -n 3 lines.txt`

or `[/data/L05]$ < lines.txt awk 'NR<=3'`

The "-n" option will not print anything unless an explicit request to print is found

The command below will print the lines starting at line 4

```
[/data/L05]$ < lines.txt  sed '1,3d'
```

The "d" deletes every line that matches the restriction

The command below will also print the lines starting at line 4

```
[/data/L05]$ < lines.txt  sed -n '1,3!p'
```

Or `[/data/L05]$ < lines.txt awk 'NR>=4'`

Or `[/data/L05]$ < lines.txt tail -n +4`

Using tail -n +4 to remove the first 3 lines and tail is much faster

The command below will start printing the lines, starting with line 1 and incrementing by 2s (1, 3, 5, …)

```
[/data/L05]$ cat -n lines.txt | sed -n '1~2p'
```

Similarly, the command below will start printing the lines, starting with line 2 and incrementing by 3s (2, 5, 8, …)

```
[/data/L05]$ cat -n lines.txt | sed -n '2~3p'
```

The command on the next line will print the lines from 1 to 3 and the lines from 5 to 19

```
[/data/L05]$ cat -n lines.txt | sed -n '1,3p;5,19p'
```

The command 'header -a' will add a header before

printing the output.

```
[/data/L05]$ seq 5 | header -a count
count
1
2
3
4
5
```

Examples:

1. To extract all the chapter headings from *Alice's Adventures in Wonderland*:

```
[/data/L05]$ grep -i chapter  alice.txt

CHAPTER I. Down the Rabbit-Hole
CHAPTER II. The Pool of Tears
CHAPTER III. A Caucus-Race and a Long Tale
CHAPTER IV. The Rabbit Sends in a Little Bill
```

CHAPTER V. Advice from a Caterpillar

CHAPTER VI. Pig and Pepper

CHAPTER VII. A Mad Tea-Party

CHAPTER VIII. The Queen's Croquet-Ground

CHAPTER IX. The Mock Turtle's Story

CHAPTER X. The Lobster Quadrille

CHAPTER XI. Who Stole the Tarts?

CHAPTER XII. Alice's Evidence

2. To print out the chapter headings which start with "The"

```
[/data/L05]$ grep -E '^CHAPTER .* The'  alice.txt
CHAPTER II. The Pool of Tears
CHAPTER IV. The Rabbit Sends in a Little Bill
CHAPTER VIII. The Queen's Croquet-Ground
CHAPTER IX. The Mock Turtle's Story
CHAPTER X. The Lobster Quadrille
```

or `[/data/L05]$ grep -E '^CHAPTER.*The'  alice.txt`

or `[/data/L05]$ grep -E '^CHAPTER (.*)\. The'  alice.txt`

3. To create a data set of all the words that start with an "a" and end with an "e", give the frequencies sorted as following :

```
|-------------+--------|
|  word       | count  |
|-------------+--------|
```

alice	403
are	73
archive	13
agree	11
anyone	5
alone	5
age	4
applicable	3
anywhere	3

|---|---|

[/data/L05]$ < alice.txt tr '[:upper:]' '[:lower:]' | grep -oE '\w+' | grep -E '^a.*e$' | sort | uniq -c | sort -nr

| awk '{print $2","$1}' | header -a 'word,count' | head -n 10 | csvlook

Or

[/data/L05]$ < alice.txt tr '[:upper:]' '[:lower:]' | grep -oE '\w{2,}' | grep -E '^a.*e$' | sort | uniq -c | sort -nr

| awk '{print $2","$1}' | header -a word,count | head | csvlook

Note:

<alice.txt grep -oE "\w+" |grep -iE "^a.*e$" |sort | uniq -c |sort -nr |head

Will output

398 Alice

63 are

13 Archive

9 agree

6 ARE

5 anyone

5 alone

5 ALICE

4 Are

4 age

So we need tr '[:upper:]' '[:lower:]' before so that we count ‘alice’, ‘Alice’ and ‘ALICE’ together.

Now say we'd like to find the emails in our document, and let's assume that emails follow the following pattern:

1) Must start with a letter
2) Email username (beginning up to @ sign) should be more than 1 character (can be letters or numbers)
3) The provider name (between @ and . can be any letter – not numbers)
4) The domain suffix (what's after the .) can be any three letters.

[/data/L05]$ grep --color -E '[a-zA-Z][a-zA-Z0-9]{1,}@[a-zA-Z]{1,}\.[a-zA-Z]{3}' file.txt

josephSmith2019@data.210 peterJobs2@data210.edu peter2Jobs@data.edu

[/data/L05]$ < file.txt tr '[:upper:]' '[:lower:]' | grep --color -E '[a-z][a-z0-9]{1,}@[a-z]{1,}\.[a-z]{3}'

Two remarks:

1. {1,} means that the length of the matched pattern needs to be 1 or more characters long
2. We needed \. to match a dot (.)

Working with CSV

we can apply SQL statements on CSV files using the command 'csvsql'. Here is a list of the cases/examples we will cover in class:

```
$ csvsql --query "SELECT * FROM tips" tips.csv |  csvlook | head -n 5
```

```
|--------+------+--------+--------+-----+--------+-------|
|  bill  | tip  | sex    | smoker | day | time   | size  |
|--------+------+--------+--------+-----+--------+-------|
|  16.99 | 1.01 | Female | 0      | Sun | Dinner | 2     |
|  10.34 | 1.66 | Male   | 0      | Sun | Dinner | 3     |
|  21.01 | 3.5  | Male   | 0      | Sun | Dinner | 3     |
|--------+------+--------+--------+-----+--------+-------|
```

• To sort the output by a column, we can use the 'ORDER BY' keyword:

```
$ csvsql --query "SELECT * FROM tips ORDER BY tip" tips.csv | csvlook | head -n 5
```

```
|------+-----+--------+--------+-----+--------+------|
|  bill | tip | sex    | smoker | day | time   | size  |
|------+-----+--------+--------+-----+--------+------|
|  3.07 | 1.0 | Female | 1      | Sat | Dinner | 1     |
|  5.75 | 1.0 | Female | 1      | Fri | Dinner | 2     |
|  7.25 | 1.0 | Female | 0      | Sat | Dinner | 1     |
|------+-----+--------+--------+-----+--------+------|
```

```
csvsql --query "SELECT * FROM tips ORDER BY bill DESC" tips.csv | csvlook | head -n 5
```

```
|--------+------+------+--------+-----+--------+-------|
|  bill  | tip  | sex  | smoker | day | time   | size  |
|--------+------+------+--------+-----+--------+-------|
|  50.81 | 10.0 | Male | 1      | Sat | Dinner | 3     |
|  48.33 | 9.0  | Male | 0      | Sat | Dinner | 4     |
|  48.27 | 6.73 | Male | 0      | Sat | Dinner | 4     |
|--------+------+------+--------+-----+--------+-------|
```

```
$ csvsql --query "SELECT smoker,sex FROM tips" tips.csv | csvlook | head -n 5
|---------+---------|
|  smoker | sex     |
|---------+---------|
|  0      | Female  |
|  0      | male    |
|  0      | male    |
|---------+---------|

$ csvsql --query "SELECT SUM(bill) FROM tips" tips.csv
SUM(bill)
4827.77

csvsql --query "SELECT AVG(bill) FROM tips" tips.csv
AVG(bill)
19.785942623

csvsql --query "SELECT MAX(bill) FROM tips" tips.csv
MAX(bill)
50.81

csvsql --query "SELECT MIN(bill) FROM tips" tips.csv
MIN(bill)
3.07

csvsql --query "SELECT DISTINCT(day) FROM tips" tips.csv
day
Sun
Sat
Thur
Fri
```

The following: count all the rows in tips.csv

```
csvsql --query "SELECT COUNT(*) FROM tips" tips.csv
COUNT(*)
244

csvsql --query "SELECT COUNT(DISTINCT(day)) FROM tips" tips.csv
COUNT(DISTINCT(day))
4

csvsql --query "SELECT COUNT(DISTINCT(day)) AS 'count of days' FROM tips" tips.csv
count of days
4
```

```
csvsql --query "SELECT * FROM tips WHERE day LIKE 's%'" tips.csv
|--------+------+--------+--------+-----+--------+-------|
|  bill  | tip  | sex    | smoker | day | time   | size  |
|--------+------+--------+--------+-----+--------+-------|
|  16.99 | 1.01 | Female | 0      | Sun | Dinner | 2     |
|  10.34 | 1.66 | Male   | 0      | Sun | Dinner | 3     |
|  21.01 | 3.5  | Male   | 0      | Sun | Dinner | 3     |
|--------+------+--------+--------+-----+--------+-------|
csvsql --query "SELECT * FROM tips WHERE day LIKE '%t'" tips.csv  | head -n 5
|--------+------+--------+--------+-----+--------+-------|
|  bill  | tip  | sex    | smoker | day | time   | size  |
|--------+------+--------+--------+-----+--------+-------|
|  20.65 | 3.35 | Male   | 0      | Sat | Dinner | 3     |
|  17.92 | 4.08 | Male   | 0      | Sat | Dinner | 2     |
|  20.29 | 2.75 | Female | 0      | Sat | Dinner | 2     |
|--------+------+--------+--------+-----+--------+-------|
```

```
csvsql --query "SELECT * FROM tips WHERE day LIKE '%i%'" tips.csv |  head -n 5
|--------+-----+--------+--------+-----+--------+-------|
|  bill  | tip | sex    | smoker | day | time   | size  |
|--------+-----+--------+--------+-----+--------+-------|
|  28.97 | 3.0 | Male   | 1      | Fri | Dinner | 2     |
|  22.49 | 3.5 | Male   | 0      | Fri | Dinner | 2     |
|  5.75  | 1.0 | Female | 1      | Fri | Dinner | 2     |
|--------+-----+--------+--------+-----+--------+-------|
csvsql --query "SELECT * FROM tips WHERE day LIKE '____'" tips.csv | head -n 5 | csvlook

|--------+------+------+--------+------+-------+-------|
|  bill  | tip  | sex  | smoker | day  | time  | size  |
|--------+------+------+--------+------+-------+-------|
|  27.2  | 4.0  | Male | 0      | Thur | Lunch | 4     |
|  22.76 | 3.0  | Male | 0      | Thur | Lunch | 2     |
|  17.29 | 2.71 | Male | 0      | Thur | Lunch | 2     |
|--------+------+------+--------+------+-------+-------|

csvsql --query "SELECT COUNT(*) FROM tips WHERE size=1" tips.csv
COUNT(*)
4


csvsql --query "SELECT MAX(bill) AS 'maximum bill' FROM tips WHERE sex='Female' and
size=1" tips.csv
maximum bill
10.07
```

csvstat Command

get the number of unique values for each column:

```
[/data/book/ch05/data]$ csvstat tips.csv --unique
  1. bill: 229
  2. tip: 123
  3. sex: 2
  4. smoker: 2
  5. day: 4
  6. time: 2
  7. size: 6
```

How to do the same thing but without 'csvstat'?

```
[/data/book/ch05/data]$ csvsql --query "SELECT COUNT(DISTINCT(sex)), COUNT(DISTINCT(size)) FROM tips" tips.csv
COUNT(DISTINCT(sex)),COUNT(DISTINCT(size))
2,6
```

More csvstat command

Show all statistics:

```
[/data/book/ch05/data]$ csvstat tips.csv

[/data/book/ch05/data]$ csvstat –c bill,tip,size,day tips.csv
```

```
[/data/book/ch05/data]$ csvstat -c sex tips.csv --unique

[/data/book/ch05/data]$ csvstat                         tips.csv --min

[/data/book/ch05/data]$ csvsql --query "SELECT MIN(bill), MIN(tip), MIN(sex) FROM tips" tips.csv

[/data/book/ch05/data]$ csvstat                         tips.csv --max

[/data/book/ch05/data]$ csvstat                         tips.csv --sum

[/data/book/ch05/data]$ csvstat                         tips.csv --mean
```

```
[/data/book/ch05/data]$ csvsql --query "SELECT AVG(bill), AVG(tip),AVG(size) FROM tips" tips.csv
```

```
[/data/book/ch05/data]$ csvstat                    tips.csv --median

[/data/book/ch05/data]$ csvstat                    tips.csv --stdev

[/data/book/ch05/data]$ csvstat                    tips.csv --unique

frequent value:
[/data/book/ch05/data]$ csvstat                    tips.csv --freq

max value length
[/data/book/ch05/data]$ csvstat                    tips.csv –len
```

Review. Make sure you run each of these commands in your Toolbox

```
[/data/L05]$cat -n file.txt | sed '1,3p'

[/data/L05]$cat -n file.txt | sed -n '1,3p'
```

The command on the next line will print the lines from 1 to 3 and the lines from 5 to 19

```
[/data/L05]$cat -n file.txt | sed -n '1,3p;5,19p'

[/data/L05]$cat -n file.txt | sed '1,3d'

[/data/L05]$cat -n file.txt | sed -n '1,3!p'

[/data/L05]$cat -n file.txt | sed -n '1~2p'

[/data/L05]$cat -n file.txt | sed -n '2~2p'

[/data/L05]$seq 100 | header -a count

[/data/L05]$tr -d 'i' < file.txt

[/data/L05]$tr -d '[iao]' < file.txt

[/data/L05]$tr -d -c '[iao]' < file.txt
```

List all the substrings that will match the regular expression (underlined) in the following command:

```
[/data/L05]$grep --color -E '[ch]{2}at' file.txt
bat chat      zat
```

if we have f3.txt as follows:

```
ccat
hat
hcat
cat
chat
hhat
```

then `[/data/L05]$ grep -E '[ch]{2}'at f3.txt`

```
ccat
hcat
chat
hhat
```

Show csv file header line:

```
[/data/book/ch05/data]$ < iris.csv header
sepal_length,sepal_width,petal_length,petal_width,species
```

delete csv file header line:

```
[/data/book/ch05/data]$ < iris.csv header -d |head -n 2
5.1,3.5,1.4,0.2,Iris-setosa
4.9,3.0,1.4,0.2,Iris-setosa
```

```
[/data/L05]$csvsql --query "SELECT * FROM n_file" n_file.txt

[/data/L05]$csvsql --query "SELECT * FROM tips ORDER BY bill" tips.csv | csvlook

[/data/L05]$csvsql --query "SELECT * FROM tips ORDER BY bill DESC" tips.csv | head | csvlook

[/data/L05]$csvsql --query "SELECT bill FROM tips ORDER BY bill DESC" tips.csv | head | csvlook

[/data/L05]$csvsql --query "SELECT * FROM tips WHERE day LIKE '%S%'" tips.csv |csvlook
```

More CSVSQL

```
vagrant@data-science-toolbox:~/book/ch07/data$ csvsql tips.csv
CREATE TABLE tips (
   bill DECIMAL NOT NULL,
   tip DECIMAL NOT NULL,
   sex VARCHAR NOT NULL,
   smoker BOOLEAN NOT NULL,
   day VARCHAR NOT NULL,
   time VARCHAR NOT NULL,
   size DECIMAL NOT NULL
);
```

The complement of columns

```
[/data/book/ch05/data]$ csvcut -c sex,day tips.csv | csvlook |head
[/data/book/ch05/data]$ csvcut -C sex,day tips.csv | csvlook |head
csvcut -d, -C'j'   1.csv >2.csv       delete column 'j'
```

csvgrep command. (FYI)

filtering on a certain pattern within a certain column

(-c is used to specify the column. -r is used to indicate that we'll use regular expressions, -i invert-matching)

The following: show all the bills of which the party size was 4 or less:

```
[/data/book/ch05/data]$ csvgrep -c size -r "[1-4]" tips.csv | csvlook
```

```
[/data/book/ch05/data]$ csvgrep –c day –r 'T' tips.csv |head
```

(-i is for inverse. The command on the next line will print the rows that DO NOT match the regular expression)

```
[/data/book/ch05/data]$ csvgrep -c size -i  -r "[1-4]" tips.csv | csvlook
```

PostgreSQL

Relational database management systems are a key component of many web sites and applications. They provide a structured way to store, organize, and access information.

PostgreSQL, or Postgres, is a relational database management system that provides an implementation of the SQL querying language. It is a popular choice for many small and large projects and has the advantage of being standards-compliant and having many advanced features like reliable transactions and concurrency without read locks.
https://blog.panoply.io/postgresql-vs.-mysql

How to login to PostgreSQL:

If deployed PostgreSQL with Docker and your postgres container is running (you can type the following command to check)

```
(base) Macbooks-MBP:Data210 macbook$ docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS                    NAMES
3cac0108cde9        postgres            "docker-entrypoint.s…"   30 minutes ago      Up 30 minutes       0.0.0.0:6552->5432/tcp   node2
c7b82e820761        postgres            "docker-entrypoint.s…"   3 weeks ago         Up 32 minutes       0.0.0.0:6551->5432/tcp   node1
```

You can login one of the node (say node1) by following:

```
(base) Macbooks-MBP:Data210 macbook$ docker exec -it node1 bash
root@c7b82e820761:/# su postgres
postgres@c7b82e820761:/$ psql
psql (12.1 (Debian 12.1-1.pgdg100+1))
Type "help" for help.

postgres=#
```

Prompt =# is waiting for you to enter PostgreSQL command.

How to quit

```
postgres=# \q
postgres@3a6c647161d0:/$ exit
exit
root@3a6c647161d0:/# exit
exit
(base) Macbooks-MBP:Data210 macbook$
```

How to show the databases;

```
postgres=# \list
postgres=# \l
```

To create a new database;

```
postgres=# CREATE DATABASE data210;
CREATE DATABASE
postgres=# \l
```

List of databases

Name	Owner	Encoding	Collate	Ctype	Access privileges
data210	postgres	UTF8	en_US.utf8	en_US.utf8	
postgres	postgres	UTF8	en_US.utf8	en_US.utf8	

```
template0 | postgres | UTF8     | en_US.utf8 | en_US.utf8 | =c/postgres          +
          |          |          |            |            | postgres=CTc/postgres
template1 | postgres | UTF8     | en_US.utf8 | en_US.utf8 | =c/postgres          +
          |          |          |            |            | postgres=CTc/postgres
(4 rows)
```

To change a database;

```
postgres=# \connect data210;
```

```
You are now connected to database "data210" as user "postgres".
```

To show the current tables;

```
data210=# \dt
No relations found.
```

To show description of a table;

```
data210=# \d employee
```

```
Did not find any relation named "employee".
```

To create a new table, we use the 'CREATE' command;

Example 01:

```
data210=# CREATE TABLE employee (
 id INT,
 f_name VARCHAR(30),
 l_name VARCHAR(30));
```

CREATE TABLE

data210=# \d employee

```
      Table "public.employee"
Column |         Type          | Modifiers
--------+-----------------------+-----------
id     | integer               |
f_name | character varying(30) |
l_name | character varying(30) |
```

Let's insert one record/row into our table;

```
data210=# INSERT INTO employee VALUES(4, 'Ben', 'Smith');
```

INSERT 0 1

Example 02:

```
CREATE TABLE countries (country_code CHAR(2) PRIMARY KEY, country_name TEXT
UNIQUE);
```

- The keyword 'UNIQUE' will enforce the constraint that the value in the column 'country_name' need to be unique.

```
INSERT INTO countries VALUES ('DE', 'Germany');
                                                                    # No error
INSERT INTO countries VALUES ('US', 'United states');
                                                                    # No error
INSERT INTO countries VALUES ('US', 'Australia');
                                                                    # Error(primary key)
INSERT INTO countries VALUES ('AU', 'Germany');
                                                                    # Error (uinique)
```

To view table content, we use the 'SELECT' statement;

```
SELECT * FROM countries;
```

```
data210=# SELECT * FROM countries;
 country_code | country_name
--------------+--------------
 DE           | Germany
 US           | United States
(2 rows)
```

To show all commands, we can use \h;

```
data210=# \h

(show line by line using “enter” key , show page by page using “space”, type ‘q’
to exit)

data210=# \h CREATE TABLE
```

To delete row(s) from a table;

```
data210=# DELETE FROM countries WHERE country_code = 'us';
DELETE 0
Value is case sensitive, ‘us’ is different from ‘US’
```

" (empty string) is not NULL;

```
CREATE TABLE cities (name TEXT NOT NULL, population INT, country_code
CHAR(2));

INSERT INTO cities VALUES('', 90, 'US');   # This will work just fine

However, this won't work:
   INSERT INTO cities (population) VALUES(10000);
```

To update row(s) from a table;

```
UPDATE cities SET name = 'Charleston' WHERE country_code = 'US' AND population = 90;
```

'CHECK' Keyword;

```
CREATE TABLE city (
name TEXT NOT NULL CHECK (name <> ''),
population INT,
country_code CHAR(2));
```

- The table above will not only enforce that the values for attribute 'name' cannot be null, it will also enforce that the values for attribute 'name' cannot be an empty string ''

More 'CHECK' Keyword;

```
CREATE TABLE employee (f_name VARCHAR(30), l_name VARCHAR(30), is_manager
CHAR(1) CHECK (is_manager in ('Y', 'N')));
```

- The table above will enforce the value for column 'is_manager' to be either 'Y' or 'N'; inserting anything else will cause an error.
  - We already have employee table in the data210 database, to run above command, we need drop it first by using:

To delete an entire table;

```
DROP TABLE employee;
```

```
data210=# drop table employee;
DROP TABLE
data210=# CREATE TABLE employee (f_name VARCHAR(30), l_name VARCHAR(30), is_manager CHAR(1)
CHECK (is_manager in ('Y', 'N')));
CREATE TABLE
data210=# INSERT INTO employee VALUES('Ben','Simith',0);
ERROR:  new row for relation "employee" violates check constraint "employee_is_manager_check
"
DETAIL:  Failing row contains (Ben, Simith, 0).
data210=# INSERT INTO employee VALUES('Ben','Simith','y');
ERROR:  new row for relation "employee" violates check constraint "employee_is_manager_check
"
DETAIL:  Failing row contains (Ben, Simith, y).
data210=# INSERT INTO employee VALUES('Ben','Simith','Y');
INSERT 0 1
data210=# SELECT * FROM employee;
 f_name | l_name | is_manager
--------+--------+------------
 Ben    | Simith | Y
(1 row)
```

To delete everything (all the rows) from a table, we do the following;

```
DELETE FROM <table_name>;
```

To delete an entire database;

```
DROP DATABASE <database_name>;
```

April 1, 2020 DATA 210

Primary Key Constraints;

A table typically has a column that contain values that uniquely identify each row in the table. This column, or columns, is called the primary key (PK) of the table and enforces the entity integrity of the table. When you specify a primary key constraint for a table, the Database Management System enforces data uniqueness automatically

- Primary keys must be unique, and NOT NULL
- A table can contain only one primary key constraint.

Foreign Key Constraints;

A foreign key (FK) is a column that is used to establish and enforce a relation between two columns in two tables. In a foreign key reference, a link is created between two tables when the column that holds the primary key value for one table is referenced by the column (foreign key) in another table. This column (from the other table) becomes a foreign key.

Customer

id	name	billaddr	shipaddr
710	A. Jones	123 Sesame St., Eureka, KS	See billing address.
730	B. Smith	456 Sesame St., Eureka, KS	See billing address.
750	C. Brown	789 Sesame St., Eureka, KS	See billing address.
770	D. White	246 Sesame St., Eureka, KS	See billing address.
820	E. Baker	135 Sesame St., Eureka, KS	See billing address.
840	F. Black	468 Sesame St., Eureka, KS	See billing address.
860	G. Scott	357 Sesame St., Eureka, KS	See billing address.
880	H. Clark	937 Sesame St., Eureka, KS	P.O. Box 9, Toledo, OH

carts

id	customer_id	cartdate
2131	710	2008-09-03 00:00:00
2461	820	2008-09-16 00:00:00
2921	730	2008-09-19 00:00:00
2937	750	2008-09-21 00:00:00
3001	750	2008-09-23 00:00:00
3002	730	2008-10-07 00:00:00
3081	880	2008-10-13 00:00:00
3197	770	2008-10-14 00:00:00
3321	860	2008-10-26 00:00:00
3937	750	2008-10-28 00:00:00

cartsitems

cart_id	item_id	qty
2131	5902	3
2131	5913	2
2461	5043	3
2461	5901	2
2921	5023	3
2921	5937	2
2937	5913	1
3001	5912	3
3001	5937	2
3002	5901	1
3081	5023	3
3081	5913	2
3197	5932	1
3321	5932	3
3937	5913	3

items

id	name	type	price
5021	thingie	widgets	9.37
5022	gadget	doodads	19.37
5023	dingus	gizmos	29.37
5041	gewgaw	widgets	5.00
5042	knickknack	doodads	10.00
5043	whatnot	gizmos	15.00
5061	bric-a-brac	widgets	2.00
5062	folderol	doodads	4.00
5063	jigger	gizmos	6.00
5901	doohickey	widgets	12.00
5902	gimmick	doodads	9.37
5903	dingbat	gizmos	9.37
5911	thingamajig	widgets	22.22
5912	thingamabob	doodads	22.22
5913	thingum	gizmos	22.22
5931	contraption	widgets	49.95
5932	whatchamacallit	doodads	59.95
5937	whatsis	gizmos	93.70

Question: why use foreign key to build the relations for tables?

Say we have a table where we keep track of the information about our website's users:

f_name	l_name	city	state	zip_code
James	Smith	Charleston	SC	29492
Michael	Smith	Chalreston	SC	29492
Robert	Smith	Charleston	SC	29492
Maria	Garcia	DeKalb	IL	60115
David	Smith	DeKalb	IL	60115
Maria	Rodriguez	Charlotte	NC	28203
...	...	...	...	...

Assuming our user base is 1000, we'll have to store 1000 * 5 (number of columns) = 5000 data points. If we split the table above into two smaller tables, and establish a primary/foreign key relation between the two tables, we'll save space and accessing/querying the tables will be much faster.

'users' Table (zip_code is a foreign key)

f_name	l_name	zip_code
James	Smith	29492
Michael	Smith	29492
Robert	Smith	29492
Maria	Garcia	60115
David	Smith	60115
Maria	Rodriguez	28203
...	...	...

'zip_code' Table (zip_code is a PK)

code	city	state
29492	Charleston	SC
60115	DeKalb	IL
28203	Charlotte	NC
...	...	...

Assuming our user base is still 1000, and the number of **unique** zip codes is 100, then the size of our both tables combined is 1000 * 3 (number of columns in 1st table) + 100 * 3 (number of columns in 2nd table)= 3300

Save 5000-3300 = 1700

Let's write PostgreSQL commands that will create the two tables above and a relation between them.

We need to create 'zip_code' table first; the attribute/column 'zip_code' needs to be a primary key;

```
CREATE TABLE zip_code (
code CHAR(5) PRIMARY KEY,
city VARCHAR(30),
state CHAR(2));
```

Now we can create the 'users' table and explicitly specify that the attribute 'zip_code' is connected to the table 'zip_code';

```
CREATE TABLE users (
f_name VARCHAR(30),
l_name VARCHAR(30),
zip_code CHAR(5) REFERENCES zip_code);
```

```
Note here that the 1st zip_code is the name of the column, and the 2nd
zip_code is the name of the table. They don't have to be the same.
```

To insert a new row into 'users' table, that 'zip_code' attribute NEEDS to map to a row in the primary key table (zip_codes), otherwise the row won't be inserted successfully. For example, assuming our tables are empty;

```
INSERT INTO users VALUES('Adam', 'Smith', '29492');
                                                    // Error, since '29492' won't be mapped to anything
INSERT INTO zip_code VALUES('29492', 'Charleston', 'SC');
                                                    // No errors
INSERT INTO users VALUES('Adam', 'Smith', '29492') ;
                                                    // Now this will work
INSERT INTO users VALUES('John', 'Smith', '29492');
                                                    // This will work too, foreign keys don't need to be unique
```

INSERT INTO zip_code VALUES('29492', 'Charleston', 'SC') ;

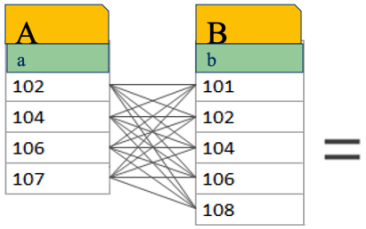
Explaining 'Cross Join';

- A *join* relates, associates, or combines two tables together to produce a single table
- Types of join: INNER JOIN, OUTER JOIN, CROSS JOIN

In PostgreSQL/MySQL, there are many commands to join tables, the easiest and most straight forward one is the 'CROSS JOIN'.

- Generate test data.
- Look for missing values.
- *every row from one table returned, joined to every row of the other table, regardless of whether they match*.

Example;



=

Results

a	b
102	101
102	102
102	104
102	106
102	108
104	101
104	102
104	104
104	106
104	108
106	101
106	102
106	104
106	106
106	108
107	101
107	102
107	104
107	106
107	108

Our example:

1. Let's rename the column "country_code" and "country_name" of table countries to "code" and "name" first.

```
data210=# ALTER TABLE countries RENAME country_code TO code;
data210=# ALTER TABLE countries RENAME country_name TO name;
```

```
SELECT * FROM countries;
 code |     name
------+--------------
 US   | United States
 DE   | Germany
(2 rows)
```

```
SELECT * FROM company;
 name  | age | salary
-------+-----+--------
 Paul  |  24 | 200000
 James |  26 | 100000
 Ben   |  26 | 150000
(3 rows)
```

```
data210=# CREATE TABLE company (name VARCHAR(30),age INT, salary INT);
data210=# INSERT INTO company VALUES ('Paul',24,200000),('James',26,
100000),('Ben',26,150000);
```

```
SELECT * FROM company CROSS JOIN countries ORDER BY company.name DESC;
name   | age | salary | code |     name
-------+-----+--------+------+--------------
 Paul  |  24 | 200000 | US   | United States
 Paul  |  24 | 200000 | DE   | Germany
 James |  26 | 100000 | US   | United States
 James |  26 | 100000 | DE   | Germany
 Ben   |  26 | 150000 | US   | United States
 Ben   |  26 | 150000 | DE   | Germany
(6 rows)
```

```
SELECT * FROM countries CROSS JOIN company;
 code |     name      | name  | age | salary
------+---------------+-------+-----+--------
 US   | United States | Paul  |  24 | 200000
 DE   | Germany       | Paul  |  24 | 200000
 US   | United States | James |  26 | 100000
 DE   | Germany       | James |  26 | 100000
 US   | United States | Ben   |  26 | 150000
 DE   | Germany       | Ben   |  26 | 150000
(6 rows)
```

Notice here that if we swap the order of the two tables, we would end up with the same resulting table, but with different column ordering.

We can specify the columns that we need to list, instead of listing all columns.

```
SELECT code, age FROM countries CROSS JOIN company;
 code | age
------+-----
 US   |  24
 DE   |  24
 US   |  26
 DE   |  26
 US   |  26
 DE   |  26
(6 rows)
```

We cannot however, do the following;

```
SELECT name, age FROM countries CROSS JOIN company;
ERROR:  column reference "name" is ambiguous
LINE 1: SELECT name, age FROM countries CROSS JOIN company;
```

... and that's because 'name' is an attribute that exists in both tables. In this case, we will need to explicitly state the table name before the column name to avoid ambiguity. Use dot notation to specify which table the column belong to, eg. countries.name, company.name.

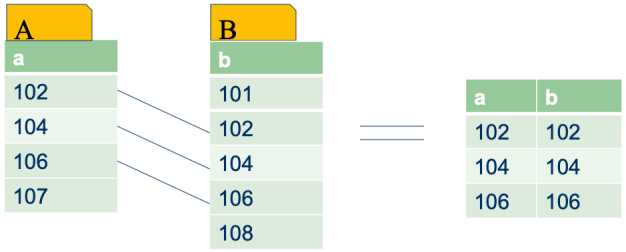
```
SELECT countries.name, age FROM countries CROSS JOIN company;

name | age
---------------+-----
 United States |  24
 Germany       |  24
 United States |  26
 Germany       |  26
 United States |  26
 Germany       |  26
(6 rows)
```

It's good practice to always specify the table of the column(s) we need to;

```
SELECT countries.name, company.age FROM countries CROSS JOIN company;
     name      | age
---------------+-----
 United States |  24
 Germany       |  24
 United States |  26
 Germany       |  26
 United States |  26
 Germany       |  26
(6 rows)
```

Inner Join

We have two tables A and B as follows,




```
csv=# CREATE TABLE A (a INT);
CREATE TABLE
csv=# CREATE TABLE B (b INT);
CREATE TABLE
csv=# INSERT INTO A VALUES (102),(104),(106),(107);
INSERT 0 4
csv=# INSERT INTO B VALUES (101),(102),(104),(106),(108);
INSERT 0 5
```

we want to combine the two tables and show A.a and B.b only for those matched rows.

```
csv=# SELECT A.a, B.b FROM A, B WHERE A.a = B.b;
  a  |  b
-----+-----
 102 | 102
 104 | 104
 106 | 106
(3 rows)
```

We can use INNER JOIN to join two tables with ON for matching condition

```
csv=# SELECT * FROM A INNER JOIN B ON A.a = B.b;
  a  |  b
-----+-----
 102 | 102
 104 | 104
 106 | 106
(3 rows)
```

- For an *inner join*, *only rows satisfying the condition in the ON clause (matched rows) are returned in the result set*.

```
shopping_cart=# SELECT
   customers.name AS customer
,  carts.id AS cart
,  items.name AS item
,  cartitems.qty
,  items.price
,  cartitems.qty * items.price  AS total
FROM
  customers INNER JOIN carts ON carts.customer_id = customers.id
    INNER JOIN cartitems  ON cartitems.cart_id = carts.id
    INNER JOIN items  ON items.id = cartitems.item_id
  ORDER BY
   customers.name
,  carts.id
,  items.name;

customer | cart |      item       | qty | price | total
---------+------+-----------------+-----+-------+--------
 A. Jones | 2131 | gimmick         |   3 |  9.37 |  28.11
 A. Jones | 2131 | thingum         |   2 | 22.22 |  44.44
 B. Smith | 2921 | dingus          |   3 | 29.37 |  88.11
 B. Smith | 2921 | whatsis         |   2 | 93.70 | 187.40
 B. Smith | 3002 | doohickey       |   1 | 12.00 |  12.00
 C. Brown | 2937 | thingum         |   1 | 22.22 |  22.22
 C. Brown | 3001 | thingamabob     |   3 | 22.22 |  66.66
 C. Brown | 3001 | whatsis         |   2 | 93.70 | 187.40
 C. Brown | 3937 | thingum         |   3 | 22.22 |  66.66
 D. White | 3197 | whatchamacallit |   1 | 59.95 |  59.95
 E. Baker | 2461 | doohickey       |   2 | 12.00 |  24.00
 E. Baker | 2461 | whatnot         |   3 | 15.00 |  45.00
 G. Scott | 3321 | whatchamacallit |   3 | 59.95 | 179.85
 H. Clark | 3081 | dingus          |   3 | 29.37 |  88.11
 H. Clark | 3081 | thingum         |   2 | 22.22 |  44.44
(15 rows)
```

Question: for each customer, show the number of items they buy and total price they pay.

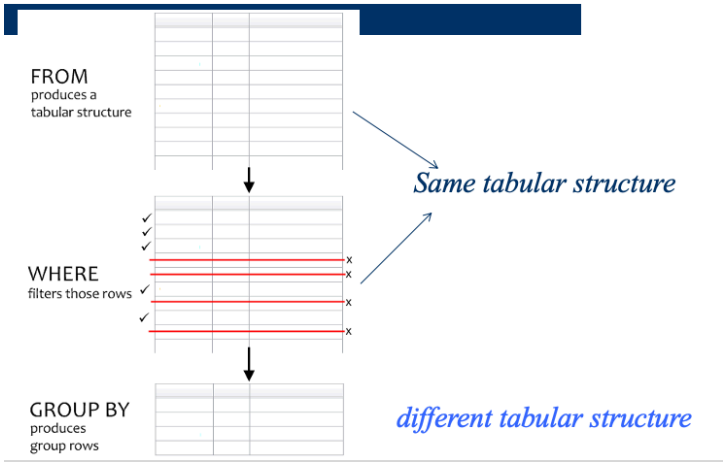
Detail Rows

customer	cart	item	qty	price	total
A. Jones	2131	gimmick	3	9.37	28.11
A. Jones	2131	thingum	2	22.22	44.44
B. Smith	2921	dingus	3	29.37	88.11
B. Smith	2921	whatsis	2	93.70	187.40
B. Smith	3002	doohickey	1	12.00	12.00
C. Brown	2937	thingum	1	22.22	22.22
C. Brown	3001	thingamabob	3	22.22	66.66
C. Brown	3001	whatsis	2	93.70	187.40
C. Brown	3937	thingum	3	22.22	66.66
D. White	3197	whatchamacallit	1	59.95	59.95
E. Baker	2461	doohickey	2	12.00	24.00
E. Baker	2461	whatnot	3	15.00	45.00
G. Scott	3321	whatchamacallit	3	59.95	179.85
H. Clark	3081	dingus	3	29.37	88.11
H. Clark	3081	thingum	2	22.22	44.44

Group Rows

customer	items	total
A. Jones	2	72.55
B. Smith	3	287.51
C. Brown	4	342.94
D. White	1	59.95
E. Baker	2	69.00
G. Scott	1	179.85
H. Clark	2	132.55

```
SELECT
   customers.name  AS customer
,  count ( items.name)  AS item
,  sum ( cartitems.qty
 * items.price ) AS total

FROM customers
    INNER JOIN carts
ON carts.customer_id = customers.id
    INNER JOIN cartitems
ON cartitems.cart_id = carts.id
    INNER JOIN items
ON items.id = cartitems.item_id

GROUP BY
   customers.name
```



'GROUP BY' clause

The PostgreSQL 'GROUP BY' clause is used in collaboration with the SELECT statement to group together those rows in a table that have identical data BY column name. We can use aggregation function (e.g SUM, AVG, MIN, MAX except COUNT) on other column to get the statistics for each group.

Examples;

```
csv=# SELECT sex, COUNT(sex) FROM tips GROUP BY sex;
sex     | count
--------+-------
 Female |    87
 Male   |   157

csv=# SELECT sex, AVG(bill) FROM tips GROUP BY sex;

 sex    |       avg
--------+------------------
```

```
 Female | 18.0568965517241
 Male   |  20.744076433121
```

You can get same results for female by

```
~/data/L08$ csvsql --query "SELECT sex, bill FROM tips WHERE sex='Female' " tips.csv |csvstat
  --mean
```

```
csv=# SELECT sex, SUM(tip) FROM tips GROUP BY sex;
sex     |  sum
  --------+--------
  Female | 246.51
  Male   | 485.07
```

```
csv=# SELECT size, count(*) FROM tips GROUP BY size;
 size | count
------+-------
    6 |     4
    4 |    37
    5 |     5
    2 |   156
    1 |     4
    3 |    38
```

```
csv=# SELECT size, count(*) FROM tips GROUP BY size ORDER BY count(*) DESC;
 size | count
------+-------
    2 |   156
    3 |    38
    4 |    37
    5 |     5
    6 |     4
    1 |     4
```

Can get same result by using [/data/L08]$ csvstat -c size tips.csv --freq

```
{ "2": 156, "3": 38, "4": 37, "5": 5, "1": 4 }
```

Show average tips on different day:

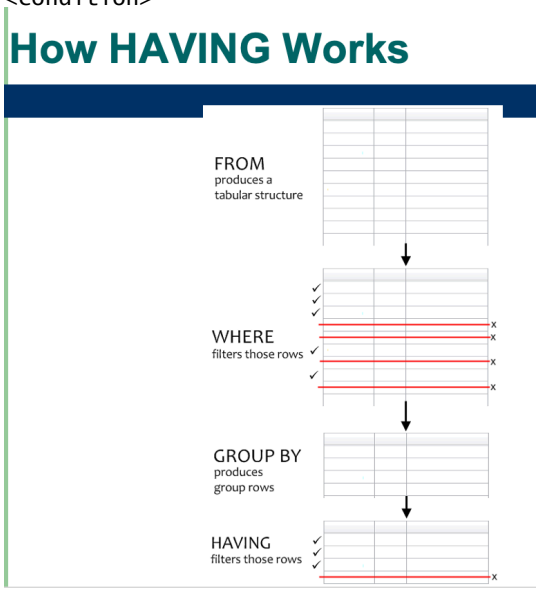
```
csv=# SELECT day ,AVG(tip) AS averageTip FROM tips GROUP BY day ORDER BY averageTip;
 day  |       avg
------+------------------
 Fri  | 2.73473684210526
 Thur | 2.77145161290323
 Sat  | 2.99310344827586
 Sun  | 3.25513157894737
```

'HAVING' & 'WHERE' clauses, when used with 'GROUP BY'

The HAVING clause allows us to pick out particular rows where the function's result meets some condition. The WHERE clause places conditions on the selected columns, whereas the HAVING clause places conditions on groups created by the GROUP BY clause.

In other words, the condition followed by the 'WHERE' clause will be applied to the rows before the grouping, while the condition that comes after the 'HAVING' clause will be applied to the rows after the grouping. Recall that the 'WHERE' clause comes before the 'GROUP BY', and that the 'HAVING' clause comes after the 'GROUP BY' clause:

```
SELECT <column_name>
<aggreationFunction(other
columns)>, FROM <table_name>
WHERE <condition>
GROUP BY <column(s)_name> HAVING
<condition>
```



Consider the table 'COMPANY' having records as follows:

```
SELECT * from COMPANY;

 id | name  | age | address    | salary
--------------------
 1 | Paul  |  32  | California  |  20000
 2 | Allen |   25  | Texas      |  15000
 3 | Teddy  |  23  | Norway  |  20000
 4 | Mark  |  25  | Rich-Mond | 65000
 5 | Allen |   27  | Texas      | 85000
 6 | Kim  |  22  | South-Hall    | 45000
 7 | Teddy  |  24  | Houston | 10000
(7 rows)
```

Following is the example, which would display record for which name count is more than 1:

```
csv=# SELECT name, COUNT(*) FROM COMPANY GROUP BY name HAVING COUNT(*) > 1;
```

This would produce the following result:

```
name       | count
---------
 Allen     | 2
 Teddy     | 2
```

However, if we execute the following command:

```
csv=# SELECT name, COUNT(*) FROM COMPANY WHERE salary > 10000 GROUP BY name HAVING COUNT(*) > 1;
```

Then we would get the following single row as our result;

```
name       | count
----------
 Allen     | 2
```

Note: We can also sort/order the result by using ORDER BY, which comes after the HAVING clause.

```
csv=# SELECT day ,AVG(tip) FROM tips WHERE time='Dinner' GROUP BY day ORDER BY AVG(tip);
 day  |       avg
------+------------------
 Fri  |             2.94
 Sat  | 2.99310344827586
 Thur |                3
  Sun  | 3.25513157894737
```

```
csv=# SELECT size,AVG(tip) from tips GROUP BY size ORDER BY AVG(tip);
 size |       avg
------+------------------
    1 |           1.4375
    2 | 2.58230769230769
    3 | 3.39315789473684
    5 |            4.028
    4 | 4.13540540540541
    6 |           5.225
```

```
csv=# SELECT size,AVG(tip)/size from tips GROUP BY size ORDER BY AVG(tip)/size;
 size |     ?column?
------+------------------
    5 |           0.8056
    6 | 0.870833333333333
    4 |  1.03385135135135
    3 |  1.13105263157895
    2 |  1.29115384615385
    1 |           1.4375
```

```
csv=# SELECT size,AVG(bill)/size from tips GROUP BY size ORDER BY AVG(bill)/size;
 size |     ?column?
------+------------------
    6 |            5.805
    5 |           6.0136
    4 | 7.15337837837838
    1 |           7.2425
    3 | 7.75921052631579
    2 | 8.22400641025642


csv=# SELECT time, AVG(bill) FROM tips GROUP BY time;
  time  |       avg
--------+------------------
 Dinner | 20.7971590909091
 Lunch  | 17.1686764705882
```

Views

Views are pseudo-tables. That is, they are not real tables, but nevertheless appear as ordinary tables to SELECT. A view can represent a subset of a real table, selecting certain columns or certain rows from an ordinary table. A view can even represent joined tables. Because views are assigned separate permissions, we can use them to restrict table access so that users see only specific rows or columns of a table.

Example;

```
csv=# CREATE VIEW company_view AS SELECT id, name, age FROM company;
```

You can list the views by:

```
csv=# \dv
```

VIEW Dependency

- If we try to drop the table 'company' after the view has been created, then we would get a dependency error. The view 'company_view' depends on the table 'company'

```
csv=# DROP TABLE company;
ERROR:  cannot drop table company because other objects depend on it
DETAIL:  view company_view depends on table company
HINT:  Use DROP ... CASCADE to drop the dependent objects too.
```

- In this case, we'll have ways to delete the table;

1) We can delete the table with CASCADE option, which would delete the table and will also delete everything that depends on it. Syntax;

```
csv=# DROP TABLE company CASCADE;
NOTICE:  drop cascades to view company_view
DROP TABLE
```

Import company.csv to database

```
csv=# CREATE TABLE company (id INT PRIMARY KEY,name VARCHAR(30),age
INT,address VARCHAR(30),salary INT);
csv=# \copy company FROM '/data210/company.csv' delimiter ',' csv header
csv=# CREATE VIEW company_view AS SELECT id, name, age FROM company;
```

2) The second option would be to first delete the view, then delete the table. Syntax

```
csv=# DROP VIEW company_view;
DROP VIEW
```

```
csv=# DROP TABLE company;
DROP TABLE
```

Indexes/Indices

- We can create an index on a column to speed the queries that are performed on that column. For example, if we know that we'll (frequently) be searching for rows that have f_name equals to something, then we can add an index constraint to the column f_name and that will make our queries much faster.
- Recall also that when we create a primary key constraint, Postgres will automatically add an index constraint on that column. The index constraint type/data-structure created on primary keys is 'btree'
- We can create an index for non primary key column and the column does not need to be unique
- We can have more than one index constraint for one table,
- We showed in class today two ways to create an index constraint on a column:

1. By not specifying the type of the index; syntax:

```
CSV=# CREATE INDEX my_index  ON company(name);
```

The script above will add an index constraint with type 'btree', we can verify that by describing the index;

```
CSV=# \d my_index
Index "public.my_index"
 Column |          Type          | Definition
--------+------------------------+------------
 name   | character varying(100) | name
   btree, for table "public.company"
```

2. By explicitly specifying the type of the index; syntax:

```
 csv=# CREATE INDEX my_other_index
csv-#  ON company USING hash(name);
CREATE INDEX
```

or

```
csv=# CREATE INDEX my_third_index ON company
USING btree(name);
CREATE INDEX
```

- The above example shows that We can have more than one index constraint for one table/column,

To delete an index:

```
csv=# DROP INDEX my_index;
DROP INDEX
```

Q01) Assume that the content of the file 'file03.txt' is the following:

Lunch
Line
Day
Dinner
Laugh

What's the output of this statement:

grep -E '^L' file03.txt | tr '\n' '.'

Ans: Lunch.Line.Laugh.

Q02) Assume that the content of the file 'file04.txt' is the

following: Nobody likes rats

I have a cat
A lot of people think that bats are birds
That's great!

What's the output of this statement:

grep -oE '[hcbr]at' file04.txt | tr -d '\n'

Ans: ratcathatbathat

Q03) What's the command that will print all the lines that start with 'Thi' AND end with 'line'; from the file 'file01.txt

Ans: grep -E '^Thi.*line$' file01.txt

Q04) What's the 'grep' command that will print the lines that contain any 5-letter substring that starts with the two letters 'Ki' and end with the letter 'e' from the file file04.txt (use the option for the extended regular expressions)

Ans: grep -E 'Ki .. e' file04.txt

12345

Q05) What's the command that will print all the lines that start with 'Thi' OR end with 'line'; from the file 'file01.txt'

Ans: grep -E ‘^Thi I \n$’ file01.txt

Q06) Assume that the content of the file file03.txt is the

following: 27.18,2.0,Female,Yes,Sat,Dinner,2

What's the output of this statement:

grep -oE 'e….' file03.txt

Ans: emale

Q07) Assume that the content of the file 'file02.txt' is the following: Lunch
Line
Day
Dinner
Laugh

What's the output of this statement:

grep -oE 'L.*n' file02.txt I tr '\n' 'j'

Ans: LunjLinj

Q08) Assume that the content of the file file03.txt is the following:

27. 18 2.0 Female Yes Sat Lunch 2

What's the output of this

statement: grep -oE '\w{3}'

file03.txt | tr -d '\n'

Ans: FemaleYesSatLun

Q09) What's the command that will print all the lines that contain the text 'Th*s'; from the file 'file01.txt'?

Ans: grep -oE ‘Th*s’ file01.txt

Q10) Assume that the content of the file 'file02.txt' is the following:

27.18 2.0 Female Yes Sat

Lunch 2 What's the output of this

statement:

grep -oE '\w{2,4}' file02.txt | tr '\n' '.'

Ans: 27.18.Fema.le.Yes.Sat.Lunc.

Q11) What's the 'csvstat' command that will show the sum for the columns 'day,sex,size' from the file 'file01.csv'

Ans: csvstat -c day, sex, size file01.csv —sum

Q12) What’s the ‘csvstat’ command that show all the statistics for all the columns from the file ‘file01.csv’

Ans: csvstat file01.csv

Q13) What's the 'csvsql' command that will display the number of rows in the file 'file03.csv'

Ans: csvsql —query “SELECT COUNT(*) FROM file03” file03.csv

Q14) What's the single 'sed' command that will display all the lines starting at line number 47; for the file file01.txt

Ans: sed ‘1,46d’ file01.txt

Q15) What's the single 'sed' command that will only display the lines from 7 to 11 for the file file02.txt

Ans: sed -n ‘7,11p’ file02.txt

Q16) What's the 'csvsql' command that will display all the 'number of unique/ distinct' values for the column 'size' from the file 'file01.csv'

Ans: csvsql —query “SELECT COUNT(DISTINCT(size)) FROM file01” file01.csv

Q17) What's the 'csvsql' command that will display all the rows but only the columns 'day' and 'sex' from the file 'file02.csv’

Ans: csvsql —query “SELECT day, sex FROM file02” file02.csv

Q18) Given that our file 'file01.txt' contains the following

text: mmm mm mmmm m

What's the output of the following commands:

a) grep -oE 'm{2,5}' file01.txt | wc -l
   Ans: mmm mm mmmm
   3

b) grep -oE 'm{2,}' file01.txt | wc -l
   Ans: mmm mm mmmm
   3

c) grep -oE 'm{2}' file01.txt | wc -l
   Ans: mm
   1

Q19) What's the 'csvsql' command that will display the maximum value for the column 'bill' from the file 'file02.csv'

Ans: csvsql —query “SELECT MAX(bill) FROM file02” file02.csv

Q20) What's the command that will delete all occurrences of the letters 'i', 'o', and 'a'; from the file 'file01.txt'

Ans: tr -d ‘[ioa]’ < file01.txt

Q21) What's the 'seq' command that will display the following:

12
11
10
09
08

Ans: seq -rw 8 12

Q22) What's the output of the following command: seq 3 | header -a Numbers

Ans: Numbers
1
2
3

Q23) Write the command that will match and print all phone number that have the following format: xxx-xxx-xxxx, where x could be any number; from the file 'file01.txt'

Ans: grep --color -E '[7-9][0-9]{2}-[0-9]{3}-[0-9]{4}' file01.txt

Q24) What's the 'csvsql' command that will display the content of the file 'file01.csv' sorted in an ascending order based on the column 'size'

Ans: csvsql —query “SELECT * FROM file01 ORDER BY size DESC”

Q25) What's the command that will delete all the letters except the letter 'b' from the file 'file02.txt' and saves the result in the file 'file01.txt'.

Ans: tr -d -c ‘b’ < file02.txt > file01.txt

DATA 210: Quiz 06
Session # __

Name:
Shefali Emmanuel

True or False (16pt)

1. Foreign keys must be unique. Ans: FALSE
2. A single table can contain only one primary key constraint. Ans: TRUE
3. A single table can contain only one foreign key constraint. Ans: FALSE
4. Primary keys must be unique. Ans: TRUE

Would the following PostgreSQL code (question 5 to 11) cause an error? If yes, please explain why (28pt)

5. CREATE TABLE company (code CHAR(5) PRIMARY KEY, city VARCHAR(30), state CHAR(2));
   CREATE TABLE employee (f_name VARCHAR(30), l_name VARCHAR(30), zip_code CHAR(5) REFERENCES company);
   INSERT INTO company VALUES('0035', 'Hattiesburg', 'MS') ;
   INSERT INTO company VALUES('0035', 'Charleston' , 'SC');

Ans: ERROR because 0035 is used 2x which violates that there can only be 1 PK per Table

6. CREATE TABLE cities (code CHAR(5) PRIMARY KEY, city VARCHAR(30), state CHAR(2));
   CREATE TABLE employee (f_name VARCHAR(30), l_name VARCHAR(30), zip_code CHAR(5) REFERENCES cites);
   INSERT INTO employee VALUES('Michael', 'Lewis', '29492') ;

Ans: ERROR because you must first INSERT INTO cities before INSERTING INTO employee

7. CREATE TABLE store (code CHAR(5) PRIMARY KEY, city VARCHAR(30), state CHAR(2));
   CREATE TABLE shopper (f_name VARCHAR(30), l_name VARCHAR(30), zip_code CHAR(5) REFERENCES place);
   INSERT INTO store VALUES('22180', 'Vienna', 'AT');
   INSERT INTO shopper VALUES('Malcolm', 'Lewis', '29492') ;

Ans: ERROR because 22180 and 29492 do not match so they can not be mapped together

8. CREATE TABLE countries (country_code char(2) PRIMARY KEY, country_name TEXT UNIQUE);
   INSERT INTO countries VALUES ('ES', 'Netherlands');
   INSERT INTO countries VALUES ('AT', 'Netherlands');

Ans: ERROR because Netherlands is used 2x

Duration: 25 minutes

9. CREATE TABLE company (name TEXT NOT NULL, employees INT, country_code CHAR(2));
   INSERT INTO company VALUES('', 9000, 'US'); // '' is two single quotations

Ans: NO ERROR

10. CREATE TABLE employee (f_name VARCHAR(30), l_name VARCHAR(30), is_manager CHAR(1) CHECK (is_manager IN ('Y', 'N')));
    INSERT INTO employee VALUES('Malcolm', 'Gladwell', 'H');

Ans: ERROR because H IS NOT 'Y' or 'N'

11. CREATE TABLE hotel (name TEXT NOT NULL CHECK (name <> 'DK'), rooms INT, country_code CHAR(2));
    INSERT INTO hotel VALUES('DK', 2000, 'AU');

Ans: ERROR because DK is not allowed

12. What's the PostgreSQL command that will delete the database 'data210' .

Ans: DROP DATABASE data210;

13. What's the PostgreSQL command that will change the column 'rooms' to the value 3, for all the rows that have the value 10 for the column 'size' in table 'db3'?

Ans: UPDATE db3 SET rooms = 3 WHERE size = 10;

14. What's the PostgreSQL command that will delete the rows in table 'tips' where the value of the column 'size' is equal to the number 3?

Ans: DELETE FROM tips WHERE size = 3;

15. What's the PostgreSQL command that will delete all the rows in table employee?

Ans: DELETE FROM employee;

16. What's the PostgreSQL command that will delete the table user?

Ans: DROP TABLE user;

17. What's the command that will describe the table 'faculty' in PostgreSQL?

Ans: \d faculty

18. What's the command that will list all the databases in PostgreSQL?

Ans: \list

19. What's the command that will make the current database 'my_database' in PostgreSQL?

Ans:\connect my_database

Assume the following: (8pt)

SELECT * FROM countries;

code	name
FR	France
DE	Germany

SELECT * FROM company;

name	age	salary
John	28	80000
Grace	26	120000

What's the output of the following commands (question 20 and 21) if there is no error? If there is an error, explain why.

20. SELECT code, age FROM countries CROSS JOIN company;

Ans: CODE | AGE
FR | 28
FR | 26
DE | 28
DE | 26

21. SELECT name, age FROM countries CROSS JOIN company;

Ans: ERROR because with name you have to do countries.name
ALSO countries doesn't contain an age column

22. Given that we have a table 'users' with the following columns: 'f_name', 'l_name', 'city', 'state', and 'zip_code', filled with 400 rows. How many data points would we save if we split our table into two tables 'users' and 'zip_codes', where 'zip_codes' has the columns 'zip_code', 'city', and 'state'. Assume that we have 30 unique zip codes.

Ans:
(user base) * (# of column in 1st) + (# of unique) * (# of column in 2nd)
400 * 5)-((400*3) + (30 * 3)) = 710 data points

Given that we have a table 'instructor' with the following columns: 'f_name' VARCHAR(30), 'l_name' VARCHAR(30),, 'department_code' CHAR(3), 'department_name' VARCHAR(30), and 'department_address VARCHAR(100), filled with 300 rows. We want to split the table into two tables: 'instructors' table and 'department' table to remove the data redundancy.

23. Which table should be created first, instructor or department, why? (2pts)

Ans: department since 'department_code needs to be a primary key

24. Please write the statement to create the two tables and using foreign key to build up the relationship between the two tables. (6pts)

```
Ans: CREATE TABLE department_code(
        code CHAR(3) PRIMARY KEY,
        department_name VARCHAR(30),
        department_address VARCHAR(100));

CREATE TABLE instructor(
        f_name VARCHAR(30),
        l_name' VARCHAR(30),
        department_code CHAR(3) REFERENCES department));
```

25. After to split the original instructor table, which has 300 rows, into the two tables shown above, how many data points would we save? Assume that we have 20 unique department code.

Ans:
(user base) * (# of column in 1st) + (# of unique) * (# of column in 2nd)
300 * 5 - ((300 * 3) + (20 * 3)) = 540 data points

Quiz Submissions - quiz07

Shefali Emmanuel (username: emmanuelsn)

Attempt 1

Written: Apr 13, 2020 10:08 AM - Apr 13, 2020 10:27 AM

Submission View

Released: Apr 11, 2020 7:03 PM

Question 1 **4 / 4 points**

Write PostgreSQL code to delete table faculty which has a dependency of a view called faculty_view . _____ ___DROP TABLE faculty CASCADE;___ ✓

Question 2 **4 / 4 points**

In PostgreSQL, write code to create an index 'my_index' for the table 'employee', for the column 'name' using 'hash'.

_____

___CREATE INDEX my_index ON employee USING hash(name);___ ✓

Question 3 **4 / 4 points**

Write PostgreSQL code to describe the index my_index. _____ ___\d my_index___ ✓

Question 4 **4 / 4 points**

In PostgreSQL, write code to create an index 'my_index_2' for the table 'employee', for the column 'name' using 'btree'.

_____

___CREATE INDEX my_index_2 ON employee USING btree(name);___ ✓

Question 5 **4 / 4 points**

Write Postgres code to delete the index 'my_index'. _____ ___DROP INDEX my_index;___ ✓

Question 6 **4 / 4 points**

Write Postgres code to list the views. _____ ___\dv___ ✓

Question 7 **4 / 4 points**

Write Postgres code to delete the view 'my_view'. _____ ___DROP VIEW my_view;___ ✓

Question 8 **4 / 4 points**

What's the syntax to create a view(my_view) with the 'f_name' and 'l_name' from the table 'user'

_____

___CREATE VIEW my_view AS SELECT f_name, l_name FROM user;___ ✓

Question 9 — 4 / 4 points

Given the table below, answer the following questions;

name	age	state	salary
Teddy	32	California	120000
Allen	32	Texas	150000
Teddy	35	Texas	140000
Mark	35	Texas	130000
Allen	32	Hawaii	180000
Allen	35	Alaska	170000

SELECT age, COUNT(state) FROM company GROUP BY age ORDER BY age;

- ○
```
age | count
-----+-------
32 |     1
35 |     1
```

- ○
```
age | count
-----+-------
32 |   2
35 |   2
```

- ○
```
age | count
-----+-------
35 |     3
32 |     3
```

- ✓ ◉
```
age | count
-----+-------
32 |   3
35 |   3
```

Question 10 — 4 / 4 points

1. When we add an index constraint to a column in our table, that column needs to be unique.

- ○ True
- ✓ ◉ False

Question 11 — 4 / 4 points

We can have more than one index constraint for one column in PostgreSQL.

- ✓ ◉ True
- ○ False

Question 12 — 4 / 4 points

We can have more than one index constraint for one table.

- ✓ ◉ True
- ○ False

Question 13 **4 / 4 points**

Giving the following:

```
SELECT * FROM employee;
 f_name      | l_name      | age
-------------+-------------+-----
 Adam        | Smith       |  30
 Ben         | Smith       |  20
 Ben         | Williams    |  30
 Ben         | Jones       |  40
Adam         | Jones       |  30
Susan        | Jones       |  20
```

What's the output of the following command?
SELECT f_name, SUM(age) FROM employee GROUP BY f_name;

- Adam 50
  Ben 70
  Susan 20

- ✓ Adam 60
  Ben 90
  Susan 20

- Ben 110
  Susan 20
  Adam 50

- Jones 90
  Smith 50
  Williams 30

Question 14 **4 / 4 points**

Giving the following:

```
SELECT * FROM employee;
 f_name      | l_name      | age
-------------+-------------+-----
 Adam        | Smith       |  30
 Ben         | Smith       |  20
 Ben         | Williams    |  30
 Ben         | Jones       |  40
Adam         | Jones       |  30
Susan        | Jones       |  20
```

What's the output of the following command?
SELECT f_name, COUNT(f_name) FROM employee GROUP BY f_name;

- Adam 3
  Ben 2
  Susan 1

- Ben 3
  Susan 2
  Adam 1

- Jones 3
  Smith 2
  Williams 1

✓ Adam 2
  Ben 3
  Susan 1

Question 15 **4 / 4 points**

Giving the following:

```
SELECT * FROM employee;
 f_name      | l_name      | age
-------------+-------------+-----
 Adam        | Smith       |  30
 Ben         | Smith       |  20
 Ben         | Williams    |  30
 Ben         | Jones       |  40
Adam         | Jones       |  30
Susan        | Jones       |  20
```

What's the output of the following command?
SELECT l_name, COUNT(age) FROM employee GROUP BY l_name;

✓ Smith 2
  Williams 1

Question 15 **4 / 4 points**

Giving the following:

```
SELECT * FROM employee;
 f_name      | l_name      | age
-------------+-------------+-----
 Adam        | Smith       |  30
 Ben         | Smith       |  20
 Ben         | Williams    |  30
 Ben         | Jones       |  40
Adam         | Jones       |  30
Susan        | Jones       |  20
```

What's the output of the following command?
SELECT l_name, COUNT(age) FROM employee GROUP BY l_name;

- ✓ (●) Smith 2
  Williams 1
  Jones 3

- ( ) Jones 3
  Smith 2
  Williams 2

- ( ) Jones 2
  Smith 2
  Williams 1

- ( ) Jones 1
  Smith 1
  Williams 2

Question 16 **4 / 4 points**

Giving the following:

```
SELECT * FROM employee;
 f_name      | l_name      | age
-------------+-------------+-----
 Adam        | Smith       |  30
 Ben         | Smith       |  20
 Ben         | Williams    |  30
 Ben         | Jones       |  40
 Adam        | Jones       |  30
 Susan       | Jones       |  20
```

What's the output of the following command?
SELECT l_name, COUNT(f_name) FROM employee GROUP BY l_name;

- ✓ (●) Jones 3
  Smith 2
  Williams 1

- ( ) Jones 2
  Smith 3
  William 1

- ( ) William 2
  Smith 2
  Jones 3

- ( ) Adam 3
  Ben 2
  Susan 1

Question 17 **4 / 4 points**

Giving the following:

```
SELECT * FROM employee;
 f_name      | l_name      | age
-------------+-------------+-----
 Adam        | Smith       |  30
 Ben         | Smith       |  20
 Ben         | Williams    |  30
 Ben         | Jones       |  40
Adam         | Jones       |  30
Susan        | Jones       |  20
```

What's the output of the following command?
SELECT f_name, AVG(age) FROM employee GROUP BY f_name;

- ✓ (●) Adam 30
  Ben 30
  Susan 20

- ( ) Adam 20
  Ben 30
  Susan 20

- ( ) Adam 30
  Ben 45
  Susan 20

- ( ) Jones 45
  Smith 25
  Williams 30

Question 18 **4 / 4 points**

Given the table below, answer the following questions;

name	age	state	salary
Teddy	32	California	120000
Allen	32	Texas	150000
Teddy	35	Texas	140000
Mark	35	Texas	130000
Allen	32	Hawaii	180000
Allen	35	Alaska	170000

```
SELECT state, count(name) FROM company WHERE salary > 140000 GROUP BY state;
```

- state | count
  --------+-------
  Hawaii | 1
  Texas | 2
  Alaska | 1

- ✓ state | count
  --------+-------
  Hawaii | 1
  Texas | 1
  Alaska | 1

- state | count
  --------+-------
  Hawaii | 1
  Texas | 3
  Alaska | 1

- state | count
  --------+-------
  California | 1
  Texas | 2
  Alaska | 2

Question 19 — 4 / 4 points

Given the table below, answer the following questions;

name	age	state	salary
Teddy	32	California	120000
Allen	32	Texas	150000
Teddy	35	Texas	140000
Mark	35	Texas	130000
Allen	32	Hawaii	180000
Allen	35	Alaska	170000

```
SELECT name, COUNT(state) FROM company WHERE age = 35 GROUP BY name;
```

✓ ◉
```
name  | count
------+-------
Allen |     1
Mark  |     1
Teddy |     1
```

○
```
name  | count
------+-------
 Allen |     1
 Teddy |     1
```

○
```
name  | count
------+-------
Mark  |     1
Teddy |     1
```

○
```
name  | count
------+-------
 Allen |     1
 Mark  |     1
 Teddy |     0
```

Question 20 — 4 / 4 points

Given the table below, answer the following questions;

name	age	state	salary
Teddy	32	California	120000
Allen	32	Texas	150000
Teddy	35	Texas	140000
Mark	35	Texas	130000
Allen	32	Hawaii	180000
Allen	35	Alaska	170000

```
SELECT state, count(name) FROM company WHERE salary < 170000 GROUP BY state;
```

✓ ◉
```
state      | count
-----------+-------
California |     1
Texas      |     3
```

Question 21 — 4 / 4 points

Given the table below, answer the following questions;

name	age	state	salary
Teddy	32	California	120000
Allen	32	Texas	150000
Teddy	35	Texas	140000
Mark	35	Texas	130000
Allen	32	Hawaii	180000
Allen	35	Alaska	170000

SELECT name, MIN(age) FROM company WHERE salary > 140000 GROUP BY name;

- name | min
  -------+-----
  Mark | 35
- name | min
  -------+-----
  Teddy | 32
  Allen | 35
- ✓ name | min
  -------+-----
  Allen | 32
- name | min
  -------+-----
  Teddy | 35
  Allen | 32

Question 22 — 4 / 4 points

Given the table below, answer the following questions;

name	age	state	salary
Teddy	32	California	120000
Allen	32	Texas	150000
Teddy	35	Texas	140000
Mark	35	Texas	130000
Allen	32	Hawaii	180000
Allen	35	Alaska	170000

```
SELECT name, COUNT(*) FROM company WHERE age = 35 GROUP BY name;
```

- ✓ name | count
  ------+-------
  Allen | 1
  Mark | 1
  Teddy | 1

Question 23 — 4 / 4 points

Given the table below, answer the following questions;

name	age	state	salary
Teddy	32	California	120000
Allen	32	Texas	150000
Teddy	35	Texas	140000
Mark	35	Texas	130000
Allen	32	Hawaii	180000
Allen	35	Alaska	170000

SELECT state, COUNT(*) FROM company GROUP BY state HAVING SUM(salary) > 200000;

- ○
```
state |count
-------
Hawaii | 1
```
- ✓ ◉
```
state | count
-------+-------
Texas |    3
```
- ○
```
state|count
-------
California |1
```
- ○
```
state|count
-------
Alaska | 2
```

Question 24 — 4 / 4 points

Given the table below, answer the following questions;

name	age	state	salary
Teddy	32	California	120000
Allen	32	Texas	150000
Teddy	35	Texas	140000
Mark	35	Texas	130000
Allen	32	Hawaii	180000
Allen	35	Alaska	170000

```
SELECT state FROM company GROUP BY state HAVING sum(salary) > 200000;
```

- ○
```
state
-------
California
```
- ○
```
state
-------
Alaska
```
- ○
```
state
-------
Hawaii
```
- ✓ ◉
```
state
-------
Texas
```

Question 25 **4 / 4 points**

Given the table below, answer the following questions;

name	age	state	salary
Teddy	32	California	120000
Allen	32	Texas	150000
Teddy	35	Texas	140000
Mark	35	Texas	130000
Allen	32	Hawaii	180000
Allen	35	Alaska	170000

```
SELECT name FROM company WHERE age = 35 GROUP BY name HAVING COUNT(*) > 1;
```

✓ name

(0 rows)

name

Teddy

name

Allen

name

Mark

Attempt Score:100 / 100 - 100 %

Overall Grade (highest attempt):100 / 100 - 100 %

Done