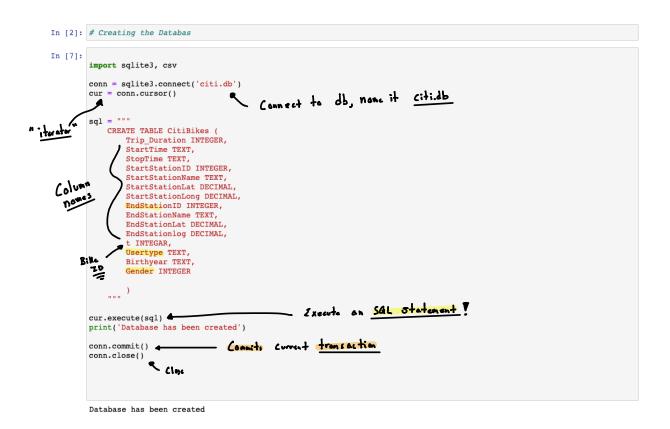


## **Creating a DB Code:**



Libikes.C3V

| A | B | Signe | Start Station | A | Start Station | B | Start Station |

## Reading CSV file into the DB

```
In [19]: # Reading the CSV file into the DB we just created
        In [20]:
                 import sqlite3, csv, pandas
                 connection = sqlite3.connect('citi.db')
cursor = connection.cursor()
                                                                                        — Creating a consor object
                 with open('citibike.csv', 'r') as file:
 next (file) -
                 no_records = 0
for row in file:
                        cursor.execute("INSERT INTO CitiBikes VALUES (?,?,?,?,?,?,?,?,?,?,?,?,?)", row.split(","))
 - to skip
                       connection.commit() Commits Current transaction
Header at Top
of COU File
                 connection.close()
print('\n{} Records Transferred'.format(no_records))
                 print("DONE!")
                 577704 Records Transferred DONE!
```

#### execute()

This routine executes an SQL statement. The SQL statement may be parameterized (i.e., placeholders instead of SQL literals). The psycopg2 module supports placeholder using %s sign

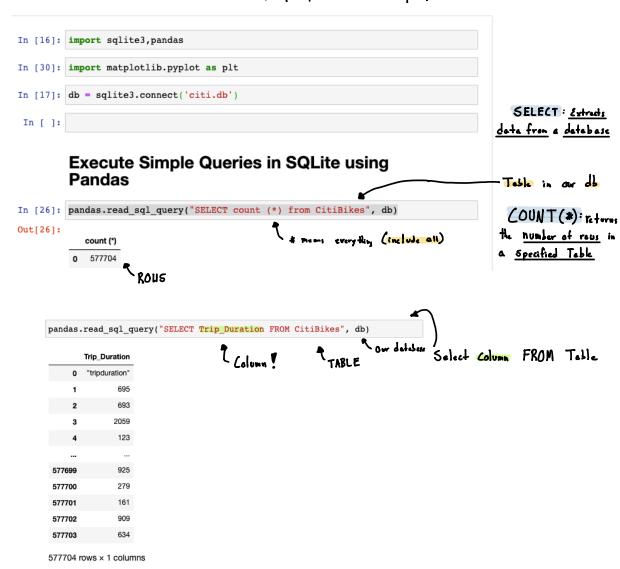
For example:cursor.execute("insert into people values (%s, %s)", (who, age))

#### **Analysis with our DB**

An **aggregate query** is a method of deriving group and subgroup data by analysis of a set of individual data entries. The term is frequently used by database developers and database administrators.

The function **COUNT**() is an aggregate function that <u>returns the number</u> of items in a group.

read\_sql\_query: Reads SQL query into a dataframe



## More with **SELECT**

# pandas.read\_sql\_query("SELECT \* FROM CitiBikes", db)

	Trip_Duration	StartTime	StopTime	StartStationID	StartStationName	StartStationLat	Star
0	"tripduration"	"starttime"	"stoptime"	"start station id"	"start station name"	"start station latitude"	
1	695	"2013-06- 01 00:00:01"	"2013-06- 01 00:11:36"	444	"Broadway & W 24 St"	40.7424	
2	693	"2013-06- 01 00:00:08"	"2013-06- 01 00:11:41"	444	"Broadway & W 24 St"	40.7424	
3	2059	"2013-06- 01 00:00:44"	"2013-06- 01 00:35:03"	406	"Hicks St & Montague St"	40.6951	
4	123	"2013-06- 01 00:01:04"	"2013-06- 01 00:03:07"	475	"E 15 St & Irving PI"	40.7352	
577699	925	"2013-06- 30 23:59:27"	"2013-07- 01 00:14:52"	509	"9 Ave & W 22 St"	40.7455	
577700	279	"2013-06- 30 23:59:36"	"2013-07- 01 00:04:15"	116	"W 17 St & 8 Ave"	40.7418	
577701	161	"2013-06- 30 23:59:33"	"2013-07- 01 00:02:14"	443	"Bedford Ave & S 9 St"	40.7085	
577702	909	"2013-06- 30 23:59:47"	"2013-07- 01 00:14:56"	509	"9 Ave & W 22 St"	40.7455	
577703	634	"2013-07- 01 00:00:00"	"2013-07- 01 00:10:34"	164	"E 47 St & 2 Ave"	40.7532	

577704 rows × 15 columns

## LIMIT

pandas.read\_sql\_query("SELECT \* FROM CitiBikes LIMIT 10", db)

	Trip_Duration	StartTime	StopTime	StartStationID	StartStationName	StartStationLat	StartStat
0	"tripduration"	"starttime"	"stoptime"	"start station id"	"start station name"	"start station latitude"	"stai lo
1	695	"2013-06- 01 00:00:01"	"2013-06- 01 00:11:36"	444	"Broadway & W 24 St"	40.7424	
2	693	"2013-06- 01 00:00:08"	"2013-06- 01 00:11:41"	444	"Broadway & W 24 St"	40.7424	
3	2059	"2013-06- 01 00:00:44"	"2013-06- 01 00:35:03"	406	"Hicks St & Montague St"	40.6951	
4	123	"2013-06- 01 00:01:04"	"2013-06- 01 00:03:07"	475	"E 15 St & Irving PI"	40.7352	
5	1521	"2013-06- 01 00:01:22"	"2013-06- 01 00:26:43"	2008	"Little West St & 1 PI"	40.7057	
6	2028	"2013-06- 01 00:01:47"	"2013-06- 01 00:35:35"	485	"W 37 St & 5 Ave"	40.7504	
7	2057	"2013-06- 01 00:02:33"	"2013-06- 01 00:36:50"	285	"Broadway & E 14 St"	40.7345	-
8	369	"2013-06- 01 00:03:29"	"2013-06- 01 00:09:38"	509	"9 Ave & W 22 St"	40.7455	
9	1829	"2013-06- 01 00:03:47"	"2013-06- 01 00:34:16"	265	"Stanton St & Chrystie St"	40.7223	
10	829	"2013-06- 01 00:04:22"	"2013-06- 01 00:18:11"	404	"9 Ave & W 14 St"	40.7406	

## return first 10 rous

#### **AVG**

```
In [43]: pandas.read_sql_query("SELECT avg(Trip_Duration) from CitiBikes", db)

Out[43]: avg(Trip_Duration)

0 1372.567903
```

## Generator

- How many rows we will get?

res = pandas.read\_sql\_query("SELECT \* FROM CitiBikes", db, chunksize = 50\_000)
res

<generator object SQLiteDatabase.\_query\_iterator at 0x7fca868c5040>

next(res) Trip\_Duration StartTime StopTime StartStationID StartStationName StartStationLat StartStationLong EndStationID E "start station name" "start station latitude" "tripduration" "starttime" "stoptime" "2013-06-01 00:00:01" "2013-06-01 00:11:36" 444 "Broadway & W 24 St" 40.7424 -73.9892 434 "2013-06- "2013-06-444 "Broadway & W 24 St" 40.7424 01 00:11:41" 00:00:08" "2013-06-01 00:00:44" "2013-06-40.6951 00:35:03 123 40.7352 -73.9876 262 00:01:04" 00:03:07 "2013-06- "2013-06-05 05 08:50:46" 09:18:52" 40.7153 -73.9602 "2013-06-05 05 08:50:51" 08:57:08" 377 40.7457 -73.9819 325 ' "2013-06- "2013-06-05 05 08:50:51" 09:05:34" "Mott St & Prince St" 40.7232 -73.9948 523 "2013-06-05 08:50:43" "2013-06-05 09:12:40" 40.7322 -74.0003 "2013-06- "2013-06-419 05 05 08:50:39" 08:57:38"

50000 rows × 15 columns

# Let's create a Helper Function!

```
# Helper Function, so our memory doesnt blow up

def Q(sql):
    res = pandas.read_sql_query(sql, db, chunksize = 100_000)
    return next(res)
```

res =	Q("SELECT	* FROM Ci	itiBikes"	')			
	Trip_Duration	StartTime	StopTime	StartStationID	StartStationName	StartStationLat	Starts
0	"tripduration"	"starttime"	"stoptime"	"start station id"	"start station name"	"start station latitude"	
1	695	"2013-06- 01 00:00:01"	"2013-06- 01 00:11:36"	444	"Broadway & W 24 St"	40.7424	
2	693	"2013-06- 01 00:00:08"	"2013-06- 01 00:11:41"	444	"Broadway & W 24 St"	40.7424	
3	2059	"2013-06- 01 00:00:44"	"2013-06- 01 00:35:03"	406	"Hicks St & Montague St"	40.6951	
4	123	"2013-06- 01 00:01:04"	"2013-06- 01 00:03:07"	475	"E 15 St & Irving PI"	40.7352	
99995	1314	"2013-06- 09 12:12:08"	"2013-06- 09 12:34:02"	439	"E 4 St & 2 Ave"	40.7263	
99996	312	"2013-06- 09 12:12:18"	"2013-06- 09 12:17:30"	476	"E 31 St & 3 Ave"	40.7439	
99997	1177	"2013-06- 09 12:12:24"	"2013-06- 09 12:32:01"	445	"E 10 St & Avenue A"	40.7274	
99998	2110	"2013-06- 09 12:12:24"	"2013-06- 09 12:47:34"	426	"West St & Chambers St"	40.7175	
99999	1058	"2013-06- 09 12:12:25"	"2013-06- 09 12:30:03"	521	"8 Ave & W 31 St N"	40.751	

100000 rows x 15 columns

## **More Simple Queries**

# Simple SQL Queries

```
# Check what Columns are in our Table
res.columns
dtype='object')
## Gender Query
res = Q("SELECT Gender, count(*) from CitiBikes GROUP BY Gender")
res
    Gender count(*)
                  O: unknown Gender
0
       0 240748
                  1: Male
       1 263492
                  2: Female
       2
         73463
2
 3 "gender"\n
      it counted the Column accident ERROR
```

# **Multiple Queries**



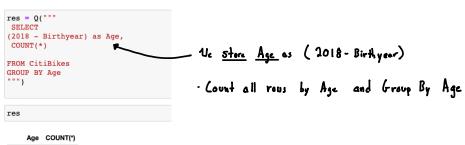
Here were selecting the Column Gender and performing COUNT all the rows and Grouping it By Gender

**GROUP BY:** to group the result-set by one or more columns

# Find Gender + Usertype:



## Let's look at age!



	Age	COUNT(*)
0	21	396
1	22	536
2	23	852
3	24	638
4	25	1504
72	105	26
73	117	28
74	118	166
75	119	66
76	2018	480642

77 rows × 2 columns

# How about Gender and Age 1:

```
res = Q("""
SELECT
Gender,(2018 - Birthyear) as Age,
COUNT(*)
FROM CitiBikes
GROUP BY Gender, Age
having Age = 40
""")
```

	Gender	Age	COUNT(*)	
0	1	40	20126	Alat of males
-	2	40	4580	

## **Order By** Keyword

```
res = Q("""

SELECT t, sum(Trip_Duration) /3600

FROM CitiBikes
GROUP BY t

ORDER BY 2
""")
```

```
o 14590
  1 15498
                           0
2 15730
                           0
  3 16049
                           0
4 16390
                           0
5789 17806
                          741
5790 17215
                          782
5791 19866
                         1058
5792 17918
                         1524
5793 15259
                         2165
```

5794 rows x 2 columns



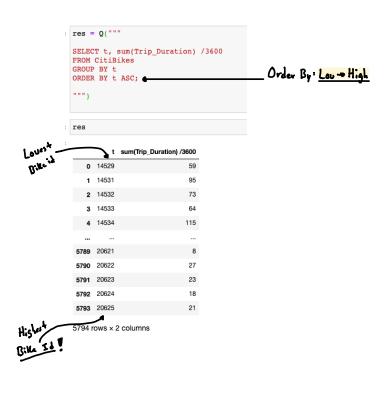
	t	sum(Trip_Duration) /3600
0	14557	0.0
1	14590	0.0
2	15498	0.0
3	15730	0.0
4	15882	0.0
5790	17806	370.0
5791	17215	391.0
5792	19866	529.0
5793	17918	762.0
5794	15259	1082.0
5705 ×	'OMO 14 5	2 columns
5/95 F	ows x z	: columns

Some Biker Lount been used.

# \* t is Bike id Number \*

order by 2 column

Order By: When there is a number its referring to Column
So 1 is column t (Bike-id) and 2 is trip-duration

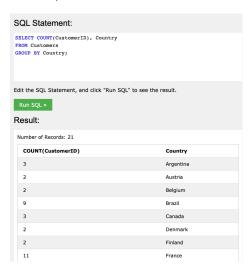


# More on **GROUP BY**: Keyword

The <u>GROUP BY</u> statement <u>groups rows that have the <u>same values</u> into <u>summary rows</u>, like "find the number of customers in each country".</u>

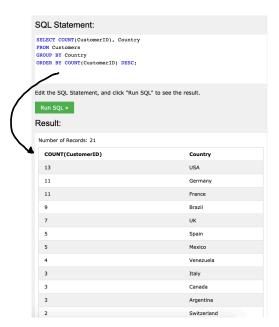
The GROUP BY statement is often used with aggregate functions (COUNT, MAX, MIN, SUM, AVG) to group the result-set by one or more columns.

# Example:



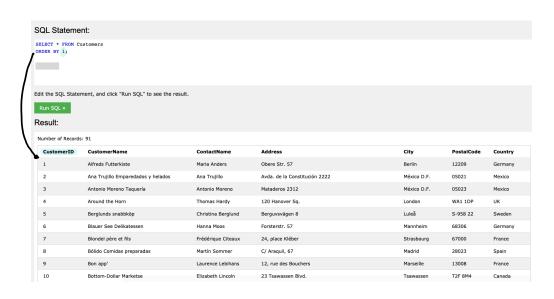
+ People for each country

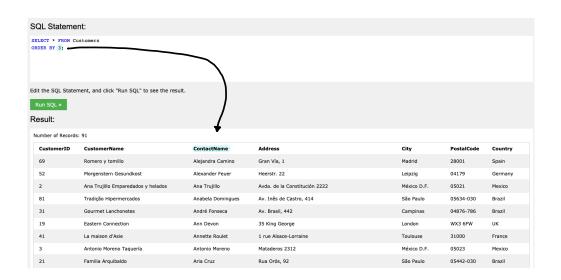
# Group By + Order By Example:



## More on **ORDER BY** Keyword:

# \* Look at the Column 7 +





# **More complex Queries**



	8	hours	hour_bucket
0	14529	59	0.0
1	14531	95	0.0
2	14532	73	0.0
3	14533	64	0.0
4	14534	115	100.0
5789	20621	8	0.0
5790	20622	27	0.0
5791	20623	23	0.0
5792	20624	18	0.0
5793	20625	21	0.0

5794 rows × 3 columns

```
res = Q (
"""

SELECT t, sum(Trip_Duration) / 3600 as hours, round(sum(Trip_Duration) / 3600 / 100) * 100 as hour_bucket
FROM CitiBikes
GROUP BY t
ORDER By hours DESC

LIMIT 12
ONLy Work

res
```

	t	hours	hour_bucket
0	15259	2165	2100.0
1	17918	1524	1500.0
2	19866	1058	1000.0
3	17215	782	700.0
4	17806	741	700.0
5	17917	619	600.0
6	18152	569	500.0
7	15043	492	400.0
8	19755	416	400.0
9	19010	382	300.0
10	17282	381	300.0
44	150/8	380	300.0

Sum time on each bike for Group By \*

## Even more complex.....

```
res = 0 ("""

SELECT hour_bucket, COUNT(*) from
(
SELECT t, sum(Trip_Duration) / 3600 as hours, round(sum(Trip_Duration) / 3600 / 100) * 100 as hour_bucket
FROM CitiBikes
GROUP BY t
)
GROUP BY hour_bucket
""")
```

	hour_bucket	COUNT(*)
0	0.0	4727
1	100.0	1006
2	200.0	40
3	300.0	12
4	400.0	2
5	500.0	1
6	600.0	1
7	700.0	2
8	1000.0	1
9	1500.0	1
10	2100.0	1

## Dropping a table using Python

You can drop a table whenever you need to, using the DROP statement of MYSQL, but you need to be very careful while deleting any existing table because the data lost will not be recovered after deleting a table.

#### Example

To drop a table from a SQLite3 database using python invoke the **execute()** method on the cursor object and pass the drop statement as a parameter to it.

```
#Connecting to sqlite
conn = sqlite3.connect('example.db')

#Creating a cursor object using the cursor() method
cursor = conn.cursor()

#Doping EMPLOYEE table if already exists
cursor.execute("DROP TABLE emp")
print("Table dropped... ")

#Commit your changes in the database
conn.commit()

#Closing the connection
conn.close()
```

## Let's clean our Data!

## **CREATE TABLE AS Statement**

```
db.execute("""

CREATE TABLE

Citi_Bike_Clean AS

SELECT

(2018 - Birthyear) AS age,

CASE WHEN gender = 0 THEN "X"

WHEN gender = 1 THEN "M"

WHEN gender = 2 THEN "F" END as sex,

*

FROM CitiBikes

WHERE age > 0

AND age <80

AND Trip_Duration < 6000
```

<sqlite3.Cursor at 0x7f8dac6b9ce0>

The **CASE** statement goes through conditions and returns a value when the first condition is met