# SQLite Citi Bikes

Project

## Creating a DB Code:

In [2]:
```
# Creating the Databas
```

In [7]:
```
import sqlite3, csv

conn = sqlite3.connect('citi.db')
cur = conn.cursor()

sql = """
    CREATE TABLE CitiBikes (
        Trip_Duration INTEGER,
        StartTime TEXT,
        StopTime TEXT,
        StartStationID INTEGER,
        StartStationName TEXT,
        StartStationLat DECIMAL,
        StartStationLong DECIMAL,
        EndStationID INTEGER,
        EndStationName TEXT,
        EndStationLat DECIMAL,
        EndStationlog DECIMAL,
        t INTEGAR,
        Usertype TEXT,
        Birthyear TEXT,
        Gender INTEGER

        )
    """

cur.execute(sql)
print('Database has been created')

conn.commit()
conn.close()
```

Connect to db, name it citi.db

"iterator"

Column Names

Bike ID

Execute an SQL statement!

Commits current transaction

Close

```
Database has been created
```

Citibikes.csv

Columns

| | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | tripduration | starttime | stoptime | start station id | start station name | start station latitude | start station longitude | end station id | end station name | end station latitude | end station longitude | bikeid | usertype | birth year | gender |
| 2 | 695 | 6/1/13 0:00 | 6/1/13 0:11 | 444 | Broadway & W 24 St | 40.7423543 | -73.98915076 | 434 | 9 Ave & W 18 St | 40.74317449 | -74.00366443 | 19678 | Subscriber | 1983 | 1 |
| 3 | 693 | 6/1/13 0:00 | 6/1/13 0:11 | 444 | Broadway & W 24 St | 40.7423543 | -73.98915076 | 434 | 9 Ave & W 18 St | 40.74317449 | -74.00366443 | 16649 | Subscriber | 1984 | 1 |
| 4 | 2059 | 6/1/13 0:00 | 6/1/13 0:35 | 406 | Hicks St & Montague S | 40.69512845 | -73.99595065 | 406 | Hicks St & Montagu | 40.69512845 | -73.99595065 | 19599 | Customer | NULL | 0 |
| 5 | 123 | 6/1/13 0:01 | 6/1/13 0:03 | 475 | E 15 St & Irving Pl | 40.73524276 | -73.98758561 | 262 | Washington Park | 40.6917823 | -73.9737299 | 16352 | Subscriber | 1960 | 1 |
| 6 | 1521 | 6/1/13 0:01 | 6/1/13 0:26 | 2008 | Little West St & 1 Pl | 40.70569254 | -74.01677685 | 310 | State St & Smith St | 40.68926942 | -73.98912867 | 15567 | Subscriber | 1983 | 1 |
| 7 | 2028 | 6/1/13 0:01 | 6/1/13 0:35 | 485 | W 37 St & 5 Ave | 40.75038009 | -73.98338988 | 406 | Hicks St & Montagu | 40.69512845 | -73.99595065 | 18445 | Customer | NULL | 0 |
| 8 | 2057 | 6/1/13 0:02 | 6/1/13 0:36 | 285 | Broadway & E 14 St | 40.73454567 | -73.99074142 | 532 | S 5 Pl & S 5 St | 40.710451 | -73.960876 | 15693 | Subscriber | 1991 | 1 |
| 9 | 369 | 6/1/13 0:03 | 6/1/13 0:09 | 509 | 9 Ave & W 22 St | 40.7454973 | -74.00197139 | 521 | 8 Ave & W 31 St N | 40.75096735 | -73.99444208 | 16100 | Subscriber | 1981 | 1 |
| 10 | 1829 | 6/1/13 0:03 | 6/1/13 0:34 | 265 | Stanton St & Chrystie S | 40.72229346 | -73.99147535 | 436 | Hancock St & Bedfo | 40.68216564 | -73.95399026 | 15234 | Subscriber | 1984 | 1 |
| 11 | 829 | 6/1/13 0:04 | 6/1/13 0:18 | 404 | 9 Ave & W 14 St | 40.7405826 | -74.00550867 | 303 | Mercer St & Spring S | 40.72362738 | -73.99949601 | 16400 | Subscriber | 1987 | 1 |
| 12 | 1316 | 6/1/13 0:04 | 6/1/13 0:26 | 423 | W 54 St & 9 Ave | 40.76584941 | -73.98690506 | 314 | Cadman Plaza West | 40.69383 | -73.990539 | 19781 | Subscriber | 1960 | 1 |
| 13 | 1456 | 6/1/13 0:04 | 6/1/13 0:28 | 502 | Henry St & Grand St | 40.714215 | -73.981346 | 532 | S 5 Pl & S 5 St | 40.710451 | -73.960876 | 18886 | Customer | NULL | 0 |
| 14 | 386 | 6/1/13 0:05 | 6/1/13 0:11 | 241 | DeKalb Ave & S Portlan | 40.68981035 | -73.97493121 | 365 | Fulton St & Grand A | 40.68223166 | -73.9614583 | 19039 | Subscriber | 1981 | 1 |
| 15 | 924 | 6/1/13 0:05 | 6/1/13 0:20 | 486 | Broadway & W 29 St | 40.7462009 | -73.98855723 | 521 | 8 Ave & W 31 St N | 40.75096735 | -73.99444208 | 16608 | Customer | NULL | 0 |
| 16 | 1233 | 6/1/13 0:06 | 6/1/13 0:27 | 527 | E 33 St & 2 Ave | 40.744023 | -73.976056 | 296 | Division St & Bower | 40.71413089 | -73.9970468 | 14761 | Subscriber | 1987 | 1 |
| 17 | 512 | 6/1/13 0:03 | 6/1/13 0:12 | 309 | Murray St & West St | 40.7149787 | -74.013012 | 300 | Shevchenko Pl & E 7 | 40.728145 | -73.990214 | 19080 | Subscriber | 1979 | 2 |
| 18 | 505 | 6/1/13 0:03 | 6/1/13 0:12 | 309 | Murray St & West St | 40.7149787 | -74.013012 | 347 | Greenwich St & W H | 40.728846 | -74.008591 | 16798 | Subscriber | 1984 | 1 |
| 19 | 833 | 6/1/13 0:07 | 6/1/13 0:21 | 503 | E 20 St & Park Ave | 40.73827428 | -73.98751968 | 503 | E 20 St & Park Ave | 40.73827428 | -73.98751968 | 19072 | Customer | NULL | 0 |
| 20 | 1818 | 6/1/13 0:08 | 6/1/13 0:38 | 257 | Lispenard St & Broadw | 40.71939226 | -74.00247214 | 500 | Broadway & W 51 S | 40.76228826 | -73.98336183 | 20349 | Customer | NULL | 0 |

# 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()

with open('citibike.csv', 'r') as file:
    no_records = 0
    for row in file:
        cursor.execute("INSERT INTO CitiBikes VALUES (?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)", row.split(","))

        connection.commit()
        no_records +=1

connection.close()
print('\n{} Records Transferred'.format(no_records))

print("DONE!")
```

next(file) - to skip Header at Top of CSV File

Creating a cursor object

Commits current transaction

```
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 returns the number of items in a group.

read_sql_query : Reads SQL query into a dataframe

```
In [16]: import sqlite3,pandas
In [30]: import matplotlib.pyplot as plt
In [17]: db = sqlite3.connect('citi.db')
In [ ]:
```

SELECT : Extracts data from a database

### Execute Simple Queries in SQLite using Pandas

Table in our db

```
In [26]: pandas.read_sql_query("SELECT count (*) from CitiBikes", db)
```

* means everything (include all)

COUNT(*): returns the number of rows in a Specified Table

Out[26]:

| | count (*) |
|---|---|
| 0 | 577704 |

ROWS

```
pandas.read_sql_query("SELECT Trip_Duration FROM CitiBikes", db)
```

Column ! TABLE our database Select Column FROM Table

| | Trip_Duration |
|---|---|
| 0 | "tripduration" |
| 1 | 695 |
| 2 | 693 |
| 3 | 2059 |
| 4 | 123 |
| ... | ... |
| 577699 | 925 |
| 577700 | 279 |
| 577701 | 161 |
| 577702 | 909 |
| 577703 | 634 |

577704 rows × 1 columns

## More with SELECT

* is keyword for all (so all columns)

```
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 Pl" | 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" | "sta 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 Pl" | 40.7352 | |
| 5 | 1521 | "2013-06-01 00:01:22" | "2013-06-01 00:26:43" | 2008 | "Little West St & 1 Pl" | 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 rows

## AVG

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

Out[43]:

| | avg(Trip_Duration) |
|---|---|
| 0 | 1372.567903 |

in seconds

## 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>
```

Call next (next 50,000 rows)

```
next(res)
```

| | Trip_Duration | StartTime | StopTime | StartStationID | StartStationName | StartStationLat | StartStationLong | EndStationID | E |
|---|---|---|---|---|---|---|---|---|---|
| 0 | "tripduration" | "starttime" | "stoptime" | "start station id" | "start station name" | "start station latitude" | "start station longitude" | "end station id" | |
| 1 | 695 | "2013-06-01 00:00:01" | "2013-06-01 00:11:36" | 444 | "Broadway & W 24 St" | 40.7424 | -73.9892 | 434 | |
| 2 | 693 | "2013-06-01 00:00:08" | "2013-06-01 00:11:41" | 444 | "Broadway & W 24 St" | 40.7424 | -73.9892 | 434 | |
| 3 | 2059 | "2013-06-01 00:00:44" | "2013-06-01 00:35:03" | 406 | "Hicks St & Montague St" | 40.6951 | -73.996 | 406 | |
| 4 | 123 | "2013-06-01 00:01:04" | "2013-06-01 00:03:07" | 475 | "E 15 St & Irving Pl" | 40.7352 | -73.9876 | 262 | |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| 49995 | 1686 | "2013-06-05 08:50:46" | "2013-06-05 09:18:52" | 539 | "Metropolitan Ave & Bedford Ave" | 40.7153 | -73.9602 | 261 | |
| 49996 | 377 | "2013-06-05 08:50:51" | "2013-06-05 08:57:08" | 472 | "E 32 St & Park Ave" | 40.7457 | -73.9819 | 325 | |
| 49997 | 883 | "2013-06-05 08:50:51" | "2013-06-05 09:05:34" | 251 | "Mott St & Prince St" | 40.7232 | -73.9948 | 523 | |
| 49998 | 1317 | "2013-06-05 08:50:43" | "2013-06-05 09:12:40" | 369 | "Washington Pl & 6 Ave" | 40.7322 | -74.0003 | 440 | |
| 49999 | 419 | "2013-06-05 08:50:39" | "2013-06-05 08:57:38" | 238 | "Bank St & Washington St" | 40.7362 | -74.0086 | 470 | |

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 CitiBikes")
res
```

| | 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 Pl" | 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 × 15 columns

## More Simple Queries

## Simple SQL Queries

```
# Check what Columns are in our Table
```

```
res.columns
```

```
Index(['Trip_Duration', 'StartTime', 'StopTime', 'StartStationID',
       'StartStationName', 'StartStationLat', 'StartStationLong',
       'EndStationID', 'EndStationName', 'EndStationLat', 'EndStationlo
g', 't',
       'Usertype', 'Birthyear', 'Gender'],
      dtype='object')
```

```
## Gender Query
```

```
res = Q("SELECT Gender, count(*) from CitiBikes GROUP BY Gender")
res
```

| | Gender | count(*) |
|---|---|---|
| 0 | 0 | 240748 |
| 1 | 1 | 263492 |
| 2 | 2 | 73463 |
| 3 | "gender"\n | 1 |

0 : unknown Gender

1 : Male

2 : Female

it counted the Colunon accident ERROR

## Multiple Queries

```
res = Q("SELECT Gender, COUNT(*) FROM CitiBikes GROUP BY Gender")
```

```
res
```

| | Gender | COUNT(*) |
|---|---|---|
| 0 | 0 | 481496 |
| 1 | 1 | 526984 |
| 2 | 2 | 146926 |

Gender:
0: Unknown
1: Male
2: Female

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:

```
res = Q("SELECT Gender, USERTYPE, COUNT(*) FROM CitiBikes GROUP BY Gender, USERTYPE ")
```

```
res
```

| | Gender | Usertype | COUNT(*) |
|---|---|---|---|
| 0 | 0 | "Customer" | 480638 |
| 1 | 0 | "Subscriber" | 858 |
| 2 | 1 | "Subscriber" | 526984 |
| 3 | 2 | "Subscriber" | 146926 |

## Let's look at age!

```
res = Q("""
 SELECT
(2018 - Birthyear) as Age,
 COUNT(*)

FROM CitiBikes
GROUP BY Age
""")
```

We store Age as (2018 - Birthyear)

- Count all rows by Age and Group By Age

```
res
```

| | 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?:

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

""")
```

```
res
```

| | Gender | Age | COUNT(*) |
|---|---|---|---|
| 0 | 1 | 40 | 20126 |
| 1 | 2 | 40 | 4580 |

A lot of males

## Order By Keyword

```
res = Q("""

SELECT t, sum(Trip_Duration) /3600
FROM CitiBikes
GROUP BY t
ORDER BY 2

""")
```

```
res
```

| | t | sum(Trip_Duration) /3600 |
|---|---|---|
| 0 | 14590 | 0 |
| 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 × 2 columns

* t is Bike id Number *

Order by 2 column

Order By: When there is a number its refering to Column
so 1 is colun t (Bike_id) and 2 is trip_duration

```
res = Q("""

SELECT t, sum(Trip_Duration) /3600
FROM CitiBikes
GROUP BY t
ORDER BY t ASC;

""")
```

Order By: Low → High

```
res
```

Lowest Bike id

| | t | sum(Trip_Duration) /3600 |
|---|---|---|
| 0 | 14529 | 59 |
| 1 | 14531 | 95 |
| 2 | 14532 | 73 |
| 3 | 14533 | 64 |
| 4 | 14534 | 115 |
| ... | ... | ... |
| 5789 | 20621 | 8 |
| 5790 | 20622 | 27 |
| 5791 | 20623 | 23 |
| 5792 | 20624 | 18 |
| 5793 | 20625 | 21 |

5794 rows × 2 columns

Highest Bike Id !

Hours in Bike Usage

```
res = Q("""

SELECT t, sum(Trip_Duration) /3600
FROM CitiBikes
GROUP BY t
ORDER BY 2

""")

res
```

| | 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 |

5795 rows × 2 columns

Some Bikes haven't been used.

# More on GROUP BY: Keyword

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

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:

SQL Statement:

```
SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country;
```

Edit the SQL Statement, and click "Run SQL" to see the result.

Run SQL »

Result:

Number of Records: 21

| COUNT(CustomerID) | Country |
|---|---|
| 3 | Argentina |
| 2 | Austria |
| 2 | Belgium |
| 9 | Brazil |
| 3 | Canada |
| 2 | Denmark |
| 2 | Finland |
| 11 | France |

← # People for each country

Group By + Order By Example:

SQL Statement:

```
SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country
ORDER BY COUNT(CustomerID) DESC;
```

Edit the SQL Statement, and click "Run SQL" to see the result.

Run SQL »

Result:

Number of Records: 21

| COUNT(CustomerID) | Country |
|---|---|
| 13 | USA |
| 11 | Germany |
| 11 | France |
| 9 | Brazil |
| 7 | UK |
| 5 | Spain |
| 5 | Mexico |
| 4 | Venezuela |
| 3 | Italy |
| 3 | Canada |
| 3 | Argentina |
| 2 | Switzerland |

## More on ORDER BY Keyword:

* Look at the Column! *

SQL Statement:

```
SELECT * FROM Customers
ORDER BY 1;
```

Edit the SQL Statement, and click "Run SQL" to see the result.

Run SQL »

Result:

Number of Records: 91

| CustomerID | CustomerName | ContactName | Address | City | PostalCode | Country |
|---|---|---|---|---|---|---|
| 1 | Alfreds Futterkiste | Maria Anders | Obere Str. 57 | Berlin | 12209 | Germany |
| 2 | Ana Trujillo Emparedados y helados | Ana Trujillo | Avda. de la Constitución 2222 | México D.F. | 05021 | Mexico |
| 3 | Antonio Moreno Taquería | Antonio Moreno | Mataderos 2312 | México D.F. | 05023 | Mexico |
| 4 | Around the Horn | Thomas Hardy | 120 Hanover Sq. | London | WA1 1DP | UK |
| 5 | Berglunds snabbköp | Christina Berglund | Berguvsvägen 8 | Luleå | S-958 22 | Sweden |
| 6 | Blauer See Delikatessen | Hanna Moos | Forsterstr. 57 | Mannheim | 68306 | Germany |
| 7 | Blondel père et fils | Frédérique Citeaux | 24, place Kléber | Strasbourg | 67000 | France |
| 8 | Bólido Comidas preparadas | Martín Sommer | C/ Araquil, 67 | Madrid | 28023 | Spain |
| 9 | Bon app' | Laurence Lebihans | 12, rue des Bouchers | Marseille | 13008 | France |
| 10 | Bottom-Dollar Marketse | Elizabeth Lincoln | 23 Tsawassen Blvd. | Tsawassen | T2F 8M4 | Canada |

SQL Statement:

```
SELECT * FROM Customers
ORDER BY 3;
```

Edit the SQL Statement, and click "Run SQL" to see the result.

Run SQL »

Result:

Number of Records: 91

| CustomerID | CustomerName | ContactName | Address | City | PostalCode | Country |
|---|---|---|---|---|---|---|
| 69 | Romero y tomillo | Alejandra Camino | Gran Vía, 1 | Madrid | 28001 | Spain |
| 52 | Morgenstern Gesundkost | Alexander Feuer | Heerstr. 22 | Leipzig | 04179 | Germany |
| 2 | Ana Trujillo Emparedados y helados | Ana Trujillo | Avda. de la Constitución 2222 | México D.F. | 05021 | Mexico |
| 81 | Tradição Hipermercados | Anabela Domingues | Av. Inês de Castro, 414 | São Paulo | 05634-030 | Brazil |
| 31 | Gourmet Lanchonetes | André Fonseca | Av. Brasil, 442 | Campinas | 04876-786 | Brazil |
| 19 | Eastern Connection | Ann Devon | 35 King George | London | WX3 6FW | UK |
| 41 | La maison d'Asie | Annette Roulet | 1 rue Alsace-Lorraine | Toulouse | 31000 | France |
| 3 | Antonio Moreno Taquería | Antonio Moreno | Mataderos 2312 | México D.F. | 05023 | Mexico |
| 21 | Familia Arquibaldo | Aria Cruz | Rua Orós, 92 | São Paulo | 05442-030 | Brazil |

# More complex Queries

Sum up trips in hours and save as hours

```
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
"""
)
```

Bike Id

```
res
```

| | t | 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

* Sum time on each bike for Group By *

```
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 want 12 rows!

```
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 |
| 11 | 15948 | 380 | 300.0 |

## Even more complex.....

```
res = Q ("""

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

""")
```

```
res
```

| | 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.

```
import sqlite3

#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


""")
```

New Column

New Column Named Sex!

```
<sqlite3.Cursor at 0x7f8dac6b9ce0>
```

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