CS 341, Fall 2024
Project 1 – CTA Database App
Due: Wednesday 9/18/2024 at 11:59pm

Overview

The goal of this project is to write a console-based Python program that inputs commands from the user and outputs data from the CTA2 L daily ridership database. SQL should be used to retrieve and compute most of the information, while Python is used to display the results and if the user chooses, to plot as well.

The Updated CTA2 L Daily Ridership Database

The updated CTA2 database consists of 5 tables: Stations, Stops, Ridership, StopDetails, and Lines. This provides information about both stations and stops in the L system.

(It makes use of foreign keys. A foreign key is a primary key stored in another table, typically used to join those tables. You may think of a foreign key as a pointer to the table where it is a primary key.

Example: Station_ID is the primary key of the Stations table, and a foreign key in the Stops and Ridership tables. This allows the Stops and Ridership tables to point to the station name in case it is needed.)

The **Stations** table denotes the stations on the CTA system. A station can have one or more stops, e.g. “Chicago/Franklin” has 2 stops.

- **Station_ID**: primary key, integer
- **Station_Name**: string

The **Stops** table denotes the stops on the CTA system. For example, “Chicago (Loop-bound)” is one of the stops at the “Chicago/Franklin” station. It is a Southbound stop and handicap-accessible (ADA).

- **Stop_ID**: primary key, integer
- **Station_ID**: the ID of the station that this stop is associated with, foreign key, integer
- **Stop_Name**: string
- **Direction**: a string that is one of N, E, S, W
- **ADA**: integer, 1 if the stop is handicap-accessible, 0 if not
- **Latitude and Longitude**: position of the stop, real numbers

The **Lines** table denotes the CTA lines, e.g. “Red” line or “Blue” line.

- **Line_ID**: primary key, integer
- **Color**: string

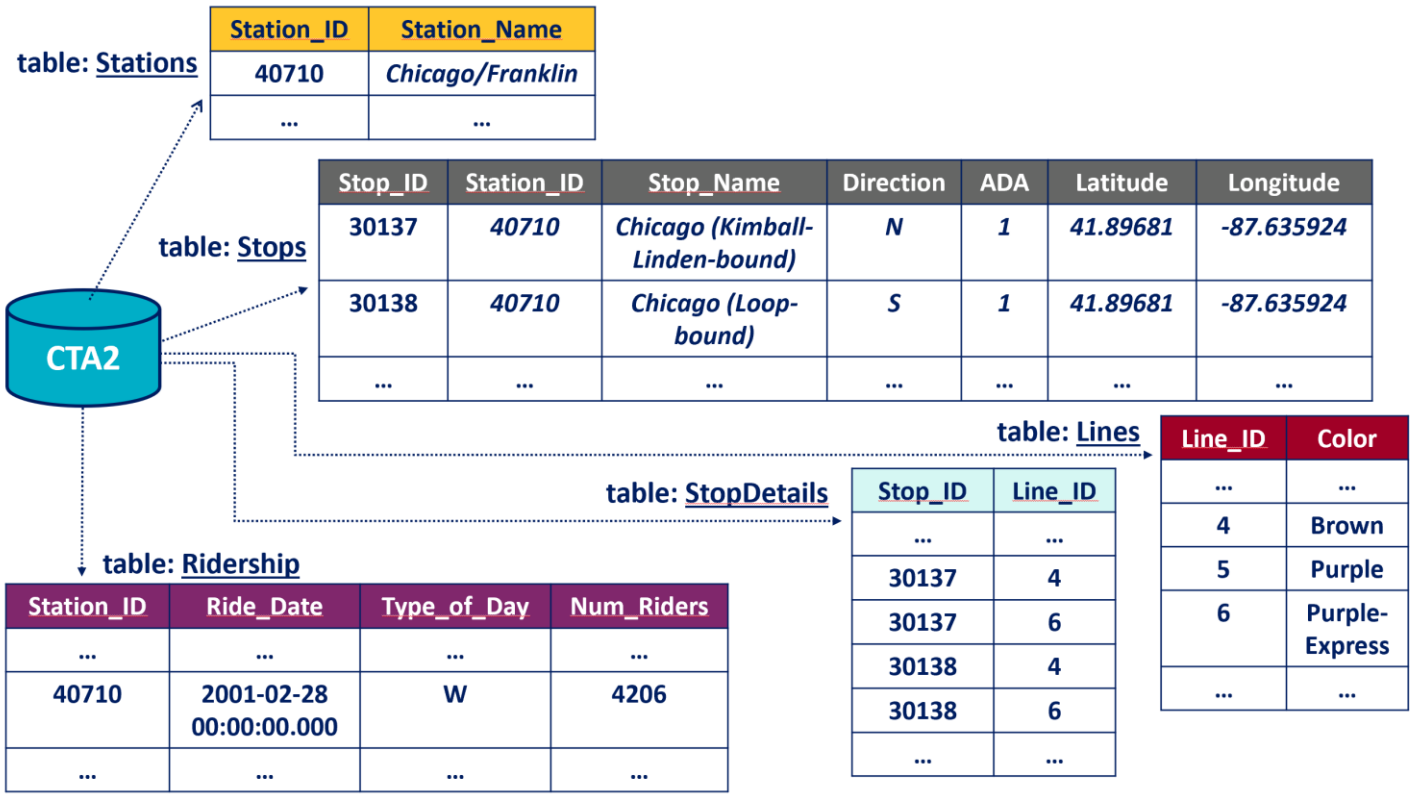
The **StopDetails** table denotes the connection between each stop and the lines. A stop may be on one or more CTA lines. As an example, “Chicago (Loop-bound)”, stop 30138, is on both the Brown and Purple-Express lines. One row in the StopDetail table denotes one unique pair (stop_id, line_id). If a stop is on multiple lines, such as “Chicago (Loop-bound)”, it will have multiple StopDetail pairs.

- **Stop_ID**: foreign key, integer
- **Line_ID**: foreign key, integer
- The pair (Stop_ID, Line_ID) forms a composite primary key

The **Ridership** table denotes how many riders went through the turnstile at this station on this date.

- **Station_ID**: foreign key, integer
- **Ride_Date**: string in format “yyyy-mm-dd hh:mm:ss.sss”
- **Type_of_Day**: string, where ‘W’ denotes a weekday, ‘A’ denotes Saturday, and ‘U’ denotes Sunday or Holiday
- **Num_Riders**: integer, total # of riders who went through the turnstile on this date
- The pair (Station_ID, Ride_Date) forms a composite primary key

Starting The Program



CS 341, Fall 2024
Project 1 – CTA Database App
Due: Wednesday 9/18/2024 at 11:59pm

The program starts by outputting some basic statistics retrieved from the database:

```
** Welcome to CTA L analysis app **

General Statistics:
  # of stations: 147
  # of stops: 302
  # of ride entries: 1,070,894
  date range: 2001-01-01 - 2021-07-31
  Total ridership: 3,377,404,512
```

You will need to match the output exactly. Note that these values may change based on the database tested against. When you submit to Gradescope for testing, we reserve the right to change the contents of the database (the schema will remain the same, but the underlying data may change to confirm you are writing a general-purpose program).

After the statistics, the program starts a command-loop, inputting string-based commands “1” - “9” or “x” to exit. All other inputs should yield an error message.

```
Please enter a command (1-9, x to exit): -1
**Error, unknown command, try again...

Please enter a command (1-9, x to exit): 0
**Error, unknown command, try again...

Please enter a command (1-9, x to exit): 10
**Error, unknown command, try again...

Please enter a command (1-9, x to exit): quit
**Error, unknown command, try again...

Please enter a command (1-9, x to exit): x
```

The program should repeat until the user inputs “x” to exit the command loop. The user can input commands in any order, and may repeat commands as often as they want.

Details on each command may be found in the sections that follow.

CS 341, Fall 2024
Project 1 – CTA Database App
Due: Wednesday 9/18/2024 at 11:59pm

Command 1

Find all the station names that match the user input. The user will be asked to input a partial station name. SQL wildcards _ and % are allowed. Output station names in ascending order. If no stations are found, print a message indicating that no stations were found.

```
Please enter a command (1-9, x to exit): 1

Enter partial station name (wildcards _ and %): %uic%
40350 : UIC-Halsted

Please enter a command (1-9, x to exit): lake
**Error, unknown command, try again...

Please enter a command (1-9, x to exit): 1

Enter partial station name (wildcards _ and %): lake
**No stations found...

Please enter a command (1-9, x to exit): 1

Enter partial station name (wildcards _ and %): 1%
40830 : 18th

Please enter a command (1-9, x to exit): 1

Enter partial station name (wildcards _ and %): %3%
41120 : 35-Bronzeville-IIT
40120 : 35th/Archer
41270 : 43rd
40910 : 63rd-Dan Ryan
40290 : Ashland/63rd
40720 : East 63rd-Cottage Grove
40940 : Halsted/63rd
40190 : Sox-35th-Dan Ryan
```

CS 341, Fall 2024
Project 1 – CTA Database App
Due: Wednesday 9/18/2024 at 11:59pm

Command 2

Given a station name, find the percentage of riders on weekdays, on Saturdays, and on Sundays/holidays for that station. The station name from the user should be an exact match. Each percentage should be calculated out of the total number of riders for that station. Display both the totals and the percentages. The totals must be computed using SQL, but the percentages can be computed using Python.

Here is one way that you can format the output in Python:

```
print("  Weekday ridership:", f"{count:,}", f"({percentage:.2f}%)")
```

In the print statement, the "f" stands for formatted output. The ":," after the count variable means that the value should be formatted with "," separators. The ":.2f" after the percentage variable means that it should output the value with 2 digits following the decimal point.

```
Please enter a command (1-9, x to exit): 2

Enter the name of the station you would like to analyze: lake
**No data found...

Please enter a command (1-9, x to exit): 2

Enter the name of the station you would like to analyze:
Clark/Lake
Percentage of ridership for the Clark/Lake station:
  Weekday ridership: 90,227,400 (90.15%)
  Saturday ridership: 5,305,557 (5.30%)
  Sunday/holiday ridership: 4,555,128 (4.55%)
  Total ridership: 100,088,085

Please enter a command (1-9, x to exit): 2

Enter the name of the station you would like to analyze: Belmont
**No data found...

Please enter a command (1-9, x to exit): 2
```

```
Enter the name of the station you would like to analyze: Belmont-
North Main
Percentage of ridership for the Belmont-North Main station:
  Weekday ridership: 56,245,014 (75.55%)
  Saturday ridership: 9,947,967 (13.36%)
  Sunday/holiday ridership: 8,259,083 (11.09%)
  Total ridership: 74,452,064

Please enter a command (1-9, x to exit): 2

Enter the name of the station you would like to analyze: Sheridan
Percentage of ridership for the Sheridan station:
  Weekday ridership: 24,787,170 (79.27%)
  Saturday ridership: 3,584,139 (11.46%)
  Sunday/holiday ridership: 2,898,941 (9.27%)
  Total ridership: 31,270,250
```

Command 3

Output the total ridership on weekdays for each station, with station names rather than the station IDs. Also show the percentages, taken out of the total ridership on weekdays for all the stations. Order the results in descending order by ridership. The totals should be computed using SQL, but percentages can be calculated in Python.

Partial output is shown below, because it is quite long.

```
Ridership on Weekdays for Each Station
Clark/Lake : 90,227,400 (3.25%)
Lake/State : 82,774,285 (2.98%)
Chicago/State : 70,350,938 (2.53%)
95th/Dan Ryan : 60,761,192 (2.19%)
Fullerton : 58,543,838 (2.11%)
. . .
King Drive : 3,175,250 (0.11%)
Cermak-McCormick Place : 2,132,001 (0.08%)
Kostner : 2,088,445 (0.08%)
Oakton-Skokie : 1,866,965 (0.07%)
Homan : 27 (0.00%)
```

CS 341, Fall 2024
Project 1 – CTA Database App
Due: Wednesday 9/18/2024 at 11:59pm

Command 4

Given a line color and direction, output all the stops for that line color in that direction. Order by stop name in ascending order. The line color and direction should be treated as case-insensitive.

There are two possible error messages here: 1) the line does not exist, or 2) the line does not run in the chosen direction. You must use the results of a SQL query to determine if either of these are the case. [HINT: Before asking the user to enter the direction, can you check using a SQL query if the line color exists? What would you expect the results to look like?]

```
Please enter a command (1-9, x to exit): 4

Enter a line color (e.g. Red or Yellow): Yellow
Enter a direction (N/S/W/E): N
Dempster-Skokie (Arrival) : direction = N (handicap accessible)
Howard (Linden & Skokie-bound) : direction = N (handicap
accessible)
Oakton-Skokie (Dempster-Skokie-bound) : direction = N (handicap
accessible)

Please enter a command (1-9, x to exit): 4

Enter a line color (e.g. Red or Yellow): gray
**No such line...

Please enter a command (1-9, x to exit): 4

Enter a line color (e.g. Red or Yellow): red
Enter a direction (N/S/W/E): W
**That line does not run in the direction chosen...

Please enter a command (1-9, x to exit): 4

Enter a line color (e.g. Red or Yellow): purple
Enter a direction (N/S/W/E): east
**That line does not run in the direction chosen...
```

CS 341, Fall 2024
Project 1 – CTA Database App
Due: Wednesday 9/18/2024 at 11:59pm

Command 5

Output the number of stops for each line color, separated by direction. Show the results in ascending order by color name, and then in ascending order by direction. Also show the percentage for each one, which is taken out of the total number of stops. The totals should be computed using SQL, but percentages can be calculated in Python.

[HINT: This will involve a multi-table join and grouping by multiple fields.]

```
Please enter a command (1-9, x to exit): 5
Number of Stops For Each Color By Direction
Blue going E : 13 (4.30%)
Blue going N : 20 (6.62%)
Blue going S : 20 (6.62%)
Blue going W : 13 (4.30%)
Brown going E : 2 (0.66%)
Brown going N : 23 (7.62%)
Brown going S : 21 (6.95%)
Brown going W : 2 (0.66%)
Green going E : 20 (6.62%)
Green going N : 12 (3.97%)
Green going S : 12 (3.97%)
Green going W : 20 (6.62%)
Orange going E : 2 (0.66%)
Orange going N : 10 (3.31%)
Orange going S : 12 (3.97%)
Orange going W : 2 (0.66%)
Pink going E : 16 (5.30%)
Pink going N : 2 (0.66%)
Pink going S : 4 (1.32%)
Pink going W : 16 (5.30%)
Purple going N : 9 (2.98%)
Purple going S : 9 (2.98%)
Purple-Express going E : 2 (0.66%)
Purple-Express going N : 19 (6.29%)
Purple-Express going S : 21 (6.95%)
Purple-Express going W : 2 (0.66%)
Red going N : 33 (10.93%)
Red going S : 33 (10.93%)
Yellow going N : 3 (0.99%)
Yellow going S : 3 (0.99%)
```

CS 341, Fall 2024
Project 1 – CTA Database App
Due: Wednesday 9/18/2024 at 11:59pm

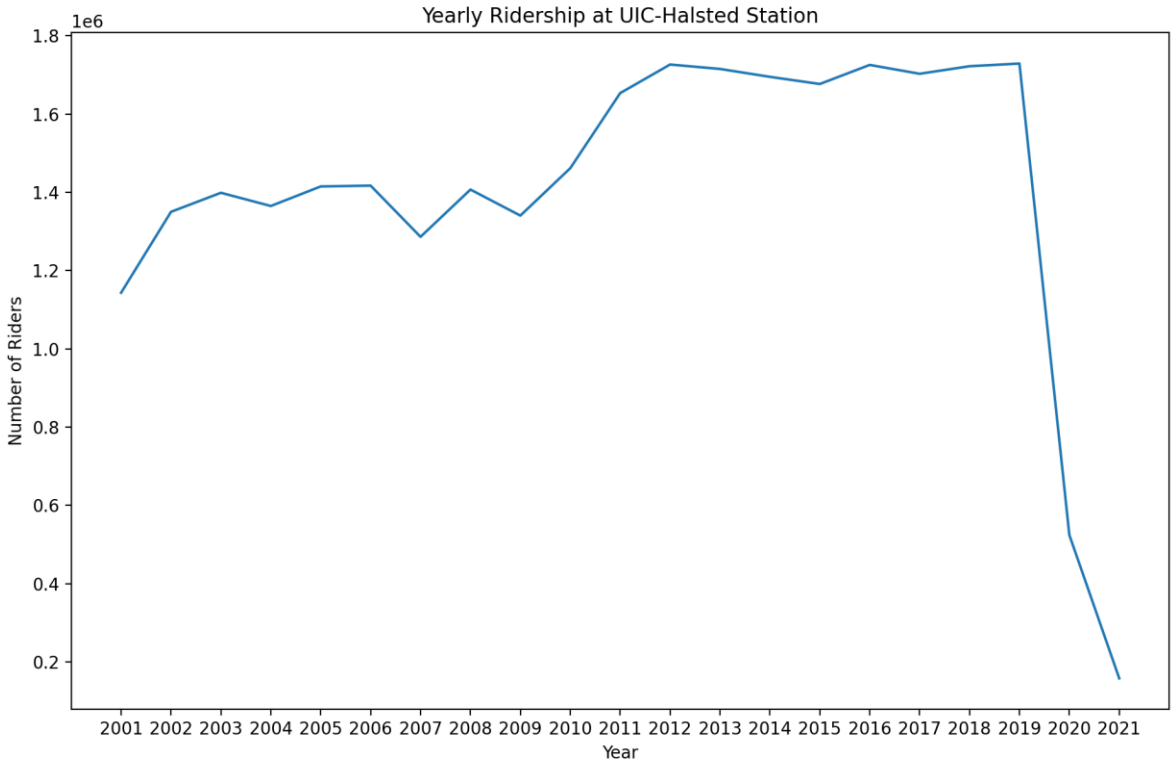
Command 6

Given a station name, output the total ridership for each year for that station, in ascending order by year. Allow the user to use wildcards _ and % for partial names. Show an error message if the station name does not exist or if multiple station names match.

After the output, the user is given the option to plot the data. Make sure the axis labels and title of the figure are set appropriately. If the user responds with any input other than "y", do not plot.

```
Please enter a command (1-9, x to exit): 6

Enter a station name (wildcards _ and %): uic
**No station found...

Please enter a command (1-9, x to exit): 6

Enter a station name (wildcards _ and %): %lake
**Multiple stations found...

Please enter a command (1-9, x to exit): 6

Enter a station name (wildcards _ and %): %uic%
Yearly Ridership at UIC-Halsted
2001 : 1,143,302
2002 : 1,350,077
2003 : 1,398,751
2004 : 1,364,987
2005 : 1,414,985
2006 : 1,417,008
2007 : 1,286,209
2008 : 1,406,969
2009 : 1,340,464
2010 : 1,461,571
2011 : 1,653,511
2012 : 1,726,553
2013 : 1,715,302
2014 : 1,695,108
2015 : 1,676,810
2016 : 1,725,557
2017 : 1,702,964
2018 : 1,722,221
```

```
2019 : 1,729,039
2020 : 524,093
2021 : 157,614

Plot? (y/n) y
```




```
Please enter a command (1-9, x to exit): 6

Enter a station name (wildcards _ and %): Sheridan
Yearly Ridership at Sheridan
2001 : 1,430,588
2002 : 1,457,916
2003 : 1,438,621
2004 : 1,372,867
2005 : 1,435,221
2006 : 1,493,220
2007 : 1,453,574
2008 : 1,602,522
2009 : 1,577,570
2010 : 1,632,450
2011 : 1,708,445
2012 : 1,771,308
```

```
2013 : 1,796,455
2014 : 1,823,423
2015 : 1,846,676
2016 : 1,862,480
2017 : 1,726,236
2018 : 1,592,654
2019 : 1,503,864
2020 : 496,168
2021 : 247,992

Plot? (y/n) n
```

Command 7

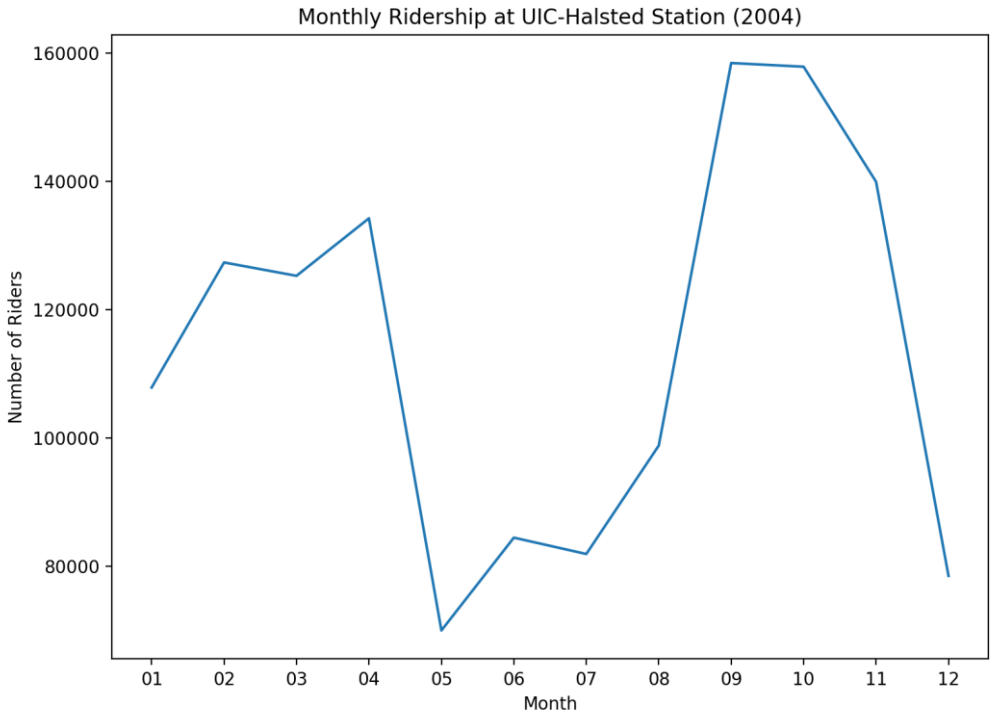
Given a station name and year, output the total ridership for each month in that year. The user should be able to enter SQL wildcards (_ and %) for the station name.

Once the station name and year have been entered, display the monthly totals. Then, give the user the option to see a plot of the results. If the user responds with “y” your program should plot as shown below (with appropriate title, legend, and axis labels). If the user responds with any other input, do not plot.

If no matching station names are found, or if multiple matching station names are found, display the corresponding error message. Note that if the user enters a year for which there is no data, no error message is necessary. The output and plot will be empty, which is sufficient.

```
Please enter a command (1-9, x to exit): 7

Enter a station name (wildcards _ and %): %uic%
Enter a year: 2004
Monthly Ridership at UIC-Halsted for 2004
01/2004 : 107,888
02/2004 : 127,382
03/2004 : 125,299
04/2004 : 134,259
05/2004 : 70,039
06/2004 : 84,500
07/2004 : 81,952
08/2004 : 98,839
09/2004 : 158,456
10/2004 : 157,882
11/2004 : 139,955
```

```
12/2004 : 78,536

Plot? (y/n) y
```




```
Please enter a command (1-9, x to exit): 7

Enter a station name (wildcards _ and %): lake
**No station found...

Please enter a command (1-9, x to exit): 7

Enter a station name (wildcards _ and %): %lake
**Multiple stations found...

Please enter a command (1-9, x to exit): 7

Enter a station name (wildcards _ and %): Sherida_
Enter a year: 2021
Monthly Ridership at Sheridan for 2021
01/2021 : 22,833
02/2021 : 24,631
03/2021 : 30,084
```

```
04/2021 : 32,519
05/2021 : 38,533
06/2021 : 45,882
07/2021 : 53,510

Plot? (y/n) n
```

Command 8

Given two station names and year, output the total ridership for each day in that year. The user should be able to enter SQL wildcards (_ and %) for each station name. Since the full output would be quite long, you should only output the first 5 days and last 5 days of data for each station. Also give the user the option to see a plot of the results. If the user responds with "y" your program should plot as shown below (with appropriate title, legend, and axis labels). If the user responds with any other input, do not plot.

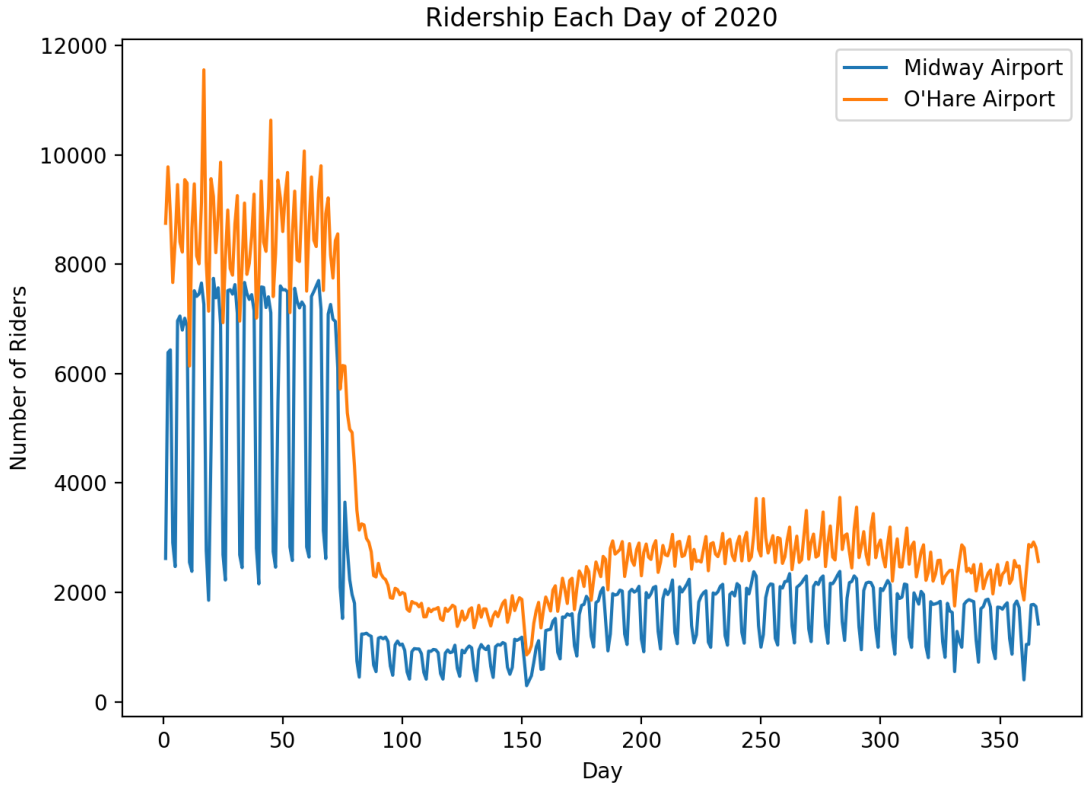
If no matching station names are found, or if multiple matching station names are found, display the corresponding error message. Note that if the user enters a year for which there is no data, no error message is necessary. The output and plot will be empty, which is sufficient.

```
Please enter a command (1-9, x to exit): 8

Year to compare against? 2020

Enter station 1 (wildcards _ and %): %midway%
**Multiple stations found...

Please enter a command (1-9, x to exit): 8

Year to compare against? 2020

Enter station 1 (wildcards _ and %): Midway %

Enter station 2 (wildcards _ and %): %o'hare%
**Multiple stations found...

Please enter a command (1-9, x to exit): 8

Year to compare against? 2020

Enter station 1 (wildcards _ and %): Midway %
```

CS 341, Fall 2024
Project 1 – CTA Database App
Due: Wednesday 9/18/2024 at 11:59pm

```
Enter station 2 (wildcards _ and %): O'Hare %
Station 1: 40930 Midway Airport
2020-01-01 2620
2020-01-02 6386
2020-01-03 6435
2020-01-04 2909
2020-01-05 2472
2020-12-27 1051
2020-12-28 1771
2020-12-29 1780
2020-12-30 1742
2020-12-31 1422
Station 2: 40890 O'Hare Airport
2020-01-01 8750
2020-01-02 9782
2020-01-03 8915
2020-01-04 7667
2020-01-05 8429
2020-12-27 2875
2020-12-28 2831
2020-12-29 2920
2020-12-30 2819
2020-12-31 2566

Plot? (y/n) y
```

CS 341, Fall 2024
Project 1 – CTA Database App
Due: Wednesday 9/18/2024 at 11:59pm




```
Please enter a command (1-9, x to exit): 8

Year to compare against? 2000

Enter station 1 (wildcards _ and %): %uic%

Enter station 2 (wildcards _ and %): Racine
Station 1: 40350 UIC-Halsted
Station 2: 40470 Racine

Plot? (y/n) n
```

CS 341, Fall 2024
Project 1 – CTA Database App
Due: Wednesday 9/18/2024 at 11:59pm

Command 9

Given a set of latitude and longitude from the user, find all stations within a mile square radius. Give the user the option to plot these stations on a map as shown below.

Check that the latitude and longitude are within the bounds of Chicago. The latitude should be between 40 and 43 degrees, and the longitude should be between -87 and -88 degrees (yes, these are quite generous boundaries). If either of these is not the case, display the appropriate message.

To find stations within a mile radius, the following information will be helpful:

- Each degree of latitude is approximately 69 miles (111 kilometers) apart (https://oceanservice.noaa.gov/facts/latitude.html).
- Using the formula: miles = cosine(degrees of latitude) · 69.17
  we find that each degree of longitude in Chicago is approximately 51 miles apart (https://www.redrockcanyonlv.org/wp-content/uploads/topographic-map-activity-6-miles-for-a-degree-of-longitude-072820.pdf).

This information should help you determine the boundaries of a square mile radius around the latitude and longitude entered. Round the numbers so that they have 3 digits after the decimal (this improves the results). You can now find all stations located within those bounds using SQL.

If the user chooses to view the plot, the locations of the stations are seen overlaying a map of Chicagoland. The map is provided as an image (.png) file in the starter code. This turns out to be surprisingly easy to do in Python. First, make sure the “chicago.png” image file is in the same folder as your Python program and the CTA2 database file. Then do the following:

```
#
# populate x and y lists with (x, y) coordinates
# note that longitude are the X values and
#      latitude are the Y values
#
x = []
y = []
.
.
.
image = plt.imread("chicago.png")
xydims = [-87.9277, -87.5569, 41.7012, 42.0868] # area covered by
the map:
plt.imshow(image, extent=xydims)
```

```
plt.title("……")

plt.plot(x, y)
#
# annotate each (x, y) coordinate with its station name:
#
for row in rows:
plt.annotate(the_station_name, (xposition, yposition))
plt.xlim([-87.9277, -87.5569])
plt.ylim([41.7012, 42.0868])
plt.show()
```

In case you're curious, here is how the image was created, and where the xydims / xlim / ylim values came from:

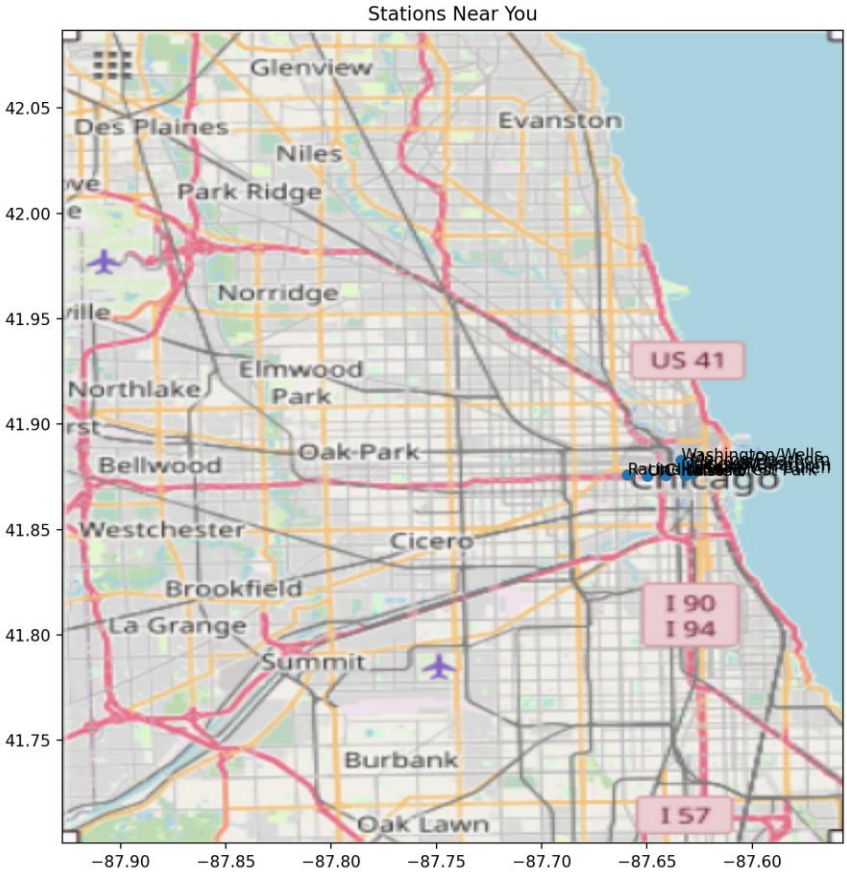
```
# Map grid from min and max position values in the CTA data:
#
# Longitude (x): -87.90422307|-87.605857
# Latitude (y): 41.722377|42.073153
#
# How to get map image?
# 1. https://www.openstreetmap.org
# 2. search for say "Chicago"
# 3. click on export in title bar
# 4. click on "manually select a different area"
# 5. resize the box
# 6. screenshot the box, save as .png file
# 7. record the 4 coordinates of the box
```

Below is the sample output for command 9.

```
Please enter a command (1-9, x to exit): 9

Enter a latitude: 41.869
Enter a longitude: -87.648

List of Stations Within a Mile
Clinton-Forest Park : (41.875539, -87.640984)
Jackson/Dearborn : (41.878183, -87.629296)
LaSalle : (41.875568, -87.631722)
LaSalle/Van Buren : (41.8768, -87.631739)
```

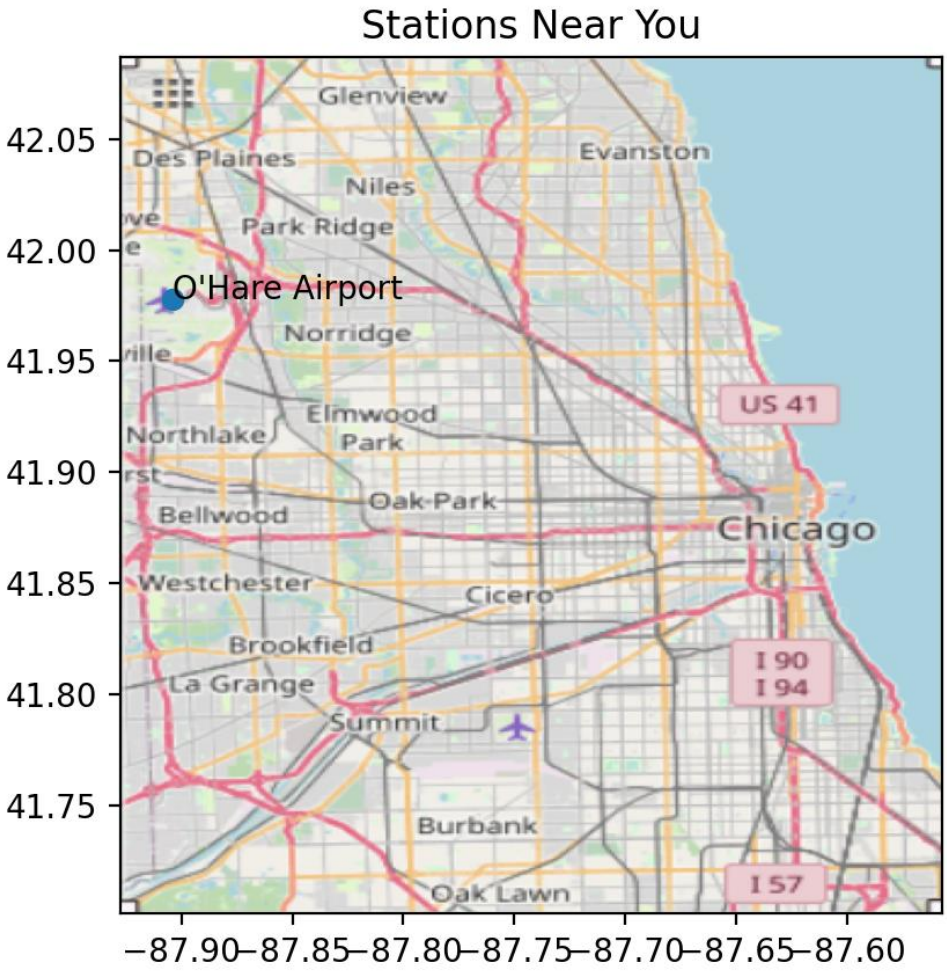
CS 341, Fall 2024
Project 1 – CTA Database App
Due: Wednesday 9/18/2024 at 11:59pm

```
Library : (41.876862, -87.628196)
Monroe/Dearborn : (41.880703, -87.629378)
Quincy/Wells : (41.878723, -87.63374)
Racine : (41.87592, -87.659458)
UIC-Halsted : (41.875474, -87.649707)
Washington/Wells : (41.882695, -87.63378)

Plot? (y/n) y
```




```
Please enter a command (1-9, x to exit): 9

Enter a latitude: 41.978
Enter a longitude: -87.9099

List of Stations Within a Mile
O'Hare Airport : (41.97766526, -87.90422307)

Plot? (y/n) y
```

CS 341, Fall 2024
Project 1 – CTA Database App
Due: Wednesday 9/18/2024 at 11:59pm




```
Please enter a command (1-9, x to exit): 9

Enter a latitude: 36
**Latitude entered is out of bounds...

Please enter a command (1-9, x to exit): 9

Enter a latitude: 41.6
Enter a longitude: 90
**Longitude entered is out of bounds...
```

CS 341, Fall 2024
Project 1 – CTA Database App
Due: Wednesday 9/18/2024 at 11:59pm

Requirements

The main requirements are that you use Python3 and SQL, and the provided SQLite database. All data retrieval and computation should be performed using explicit, string-based SQL queries executed via the sqlite3 package; no tool-generated code is allowed (e.g. you cannot use SQLAlchemy). This implies that SQL must be used to do the vast majority of computation, e.g. all searching and summing and sorting. It is not valid to load all the data into objects and then write code to do the searching and sorting yourself. For plotting, use the matplotlib package. (Some of the expected output contains percentages; computation of percentages in this case is best done using Python, so the use of SQL is not required.)

It is also expected that you use good programming practices, i.e. functions, comments, consistent spacing, etc. In a 3xx class this should be obvious and not require further explanation. But to be clear, if you submit a program with no functions and no comments, you will be significantly penalized --- in fact you can expect a score of 0, even if the program produces the correct results. How many functions? How many comments? You can decide. Make reasonable decisions and you will not be penalized.

Submission

Login to Gradescope.com and look for the assignment “Project 01”.

Submit just your main.py file under “Project 01”. **You have unlimited submissions. Keep in mind we grade your last submission unless you select an earlier submission for grading. If you do choose to activate an earlier submission, you must do so before the deadline.**

The score reported on Gradescope is only part of your final score (50%). After the project is due, the TAs will manually review the programs for style (10%) and adherence to requirements (0-100%), and then manually run the programs to check the required plotting functionality (40%).

Suggestion: when you fail a test on Gradescope, we show your output, the correct output, and the difference between the two (as computed by Linux’s diff utility). Please study the output carefully to see where your output differs. If there are lots of differences, or you can’t see the difference, here’s a good tip:

1. Browse to https://www.diffchecker.com/
2. Copy your output as given by Gradescope and paste into the left window
3. Copy the correct output as given by Gradescope and paste into the right window
4. Click the “Find Difference” button

You'll get a visual representation of the differences. Modify your program to produce the required output, and resubmit.

In terms of grading, note that we expect all submissions to compile, run, and pass at least some of the test cases; do not expect partial credit with regards to correctness unless your program compiles and runs.

Late submissions for this project are allowed. You may turn in a project up to 3 days (72 hours) late, and will receive the following penalty on your total score:

Submission made **up to 24 hours late**: **10** point deduction

Submission made **24-48 hours late**: **20** point deduction

Submission made **48-72 hours late**: **30** point deduction

No submissions will be accepted after 72 hours.

Academic Integrity

All work is to be done individually — group work is not allowed.

While we encourage you to talk to your peers and learn from them, this interaction must be superficial with regards to all work submitted for grading. This means you cannot work in teams, you cannot work side-by-side, you cannot submit someone else's work (partial or complete) as your own, etc. The University's policy is available here: https://dos.uic.edu/community-standards/

In particular, note that **you are guilty of academic dishonesty if you extend or receive any kind of unauthorized assistance**. Absolutely no transfer of program code between students is permitted (paper or electronic), and you may not solicit code from family, friends, or online forums (e.g. you cannot download answers from Chegg). Other examples of academic dishonesty include emailing your program to another student, sharing your screen so that another student may copy your work, copying-pasting code from the internet, working together in a group, and allowing a tutor, TA, or another individual to write an answer for you.

Academic dishonesty is unacceptable, and penalties range from a letter grade drop to expulsion from the university; cases are handled via the official student conduct process described at the link above.