

SOLVE IO Co-operative Education Project



Fall 2022

Eliza Clamor

Supervisor: Professor Jean-Claude Ngatchou

Introduction	2
Solve Data IO	2
My Job as a Data Analyst	2
Work infrastructure and Points of Learning	2
Example: Metric Tracker made for a Client	3
My Project - A Scheduling Algorithm using Data from a Database	4
Introduction	4
My Solution	4
Relating to my Work at Solve	4
The Interaction between Python and mySQL	5
The Database	5
The Algorithm	5
Specifications	6
Product Features	6
Software Requirements and Prerequisites	6
Performance Details	6
Limitations	6
Program Run-through	7
Creating a Connection to MySQL and Creating a Database	7
Populating the Database	7
Querying the Database	9
Using the Data to create a schedule	9
Creating the Schedule	10
Notes	11
Code	12
project.py	12
database.py	13
queries.py	15
print_schedule.py	16

Introduction

Solve Data IO

601 W 26th St, 3rd Floor, New York, NY 10001-1101

Solve Data IO is a start-up that encourages e-commerce businesses to own and take control of their own data. This allows clients to collect and have access to first party data (when a company collects and owns information directly from the customers) which, with the help of the SOLVE data analysts, can use this to make smarter business decisions. The company provides its services to e-commerce businesses in many different countries such as New Zealand, Australia, and Canada. The key seller about the company is its efficiency in using queries. Many alike businesses want the same analysis of data and so one of the founders, Neil Capel posed the question why should you write the same query when you can reuse it? Implementing this method and narrowing down the target audience allowed the Solve team to utilize this thinking, dramatically decreasing the work load.

My Job as a Data Analyst

My responsibilities consisted of creating custom queries for different clients pertaining their particular needs. I worked with the Strategy and Analytics (S&A) team to query and understand the data further than just the numbers but how they related to the business as a whole. There is 6 people in my team, 5 people in the engineering team and 2 people in the product team. We use the Presto SQL query engine. By creating queries that work in conjunction with excel manipulation, I was able to produce graphs that better visualized the results. From this I could present to client's and improve their understanding of their own statistics.

Work infrastructure and Points of Learning

I currently work in our own stacks which implements Presto as the query engine. Beyond the Computer Science 306: Database Design (CS306) class, I have learnt how to more efficiently use other functions such as functions in relation to JavaScript Object Notation (JSON) fields and array-related functions. In SQL, arrays cannot exist in a datapoint unless stored in a JSON file, otherwise they are single data points.

Example: Metric Tracker made for a Client

Using the following data from a query, I was able to create a Metric tracker which was easy on the eye and simple to understand.

week_beginning	total_revenue	new_revenue	sign_ups	fb_ad_spend	google_ad_spend
2022-11-28	36577.45	1389.35	97	702	550
2022-11-21	36590.06	1212.18	105	822	874
2022-11-14	38463.38	2025.29	118	962	1062
2022-11-07	35317.34	1922.16	119	1242	1436
2022-10-31	38013.41	2046.66	122	1624	1842

This Month							
	ex gst	Target	MTD	Delta	Progress	Weekly Target	Daily RRR
Days		31	7	24	23%		
Revenue	\$180,000	\$207,000	\$0	\$207,000	0%	44116	\$8,625
Signups		200	0	200	0%	56	8.33
Last 5 Weeks							
Start Mon	24 – 30 Oct	31 – 06 Oct	07 – 13 Nov	14 – 20 Nov	21 – 27 Nov		
	Previous 4 Weeks				Last Week	Week Target	LW % Target
Periodic Revenue	\$34,577	\$35,967	\$33,395	\$36,438	\$35,378	\$30,000	118%
New Revenue	\$3,311	\$2,047	\$1,922	\$2,025	\$1,212	\$2,000	61%
Total Revenue	\$37,888	\$38,013	\$35,317	\$38,463	\$36,590	\$44,116	83%
Signups	47	37	27	34	19	56	34%
FB Spend	1094.68	762.84	620.5	459.58	420.42		
Google	1878.32	962.14	747.79	551.06	458.14		
Other							
Marketing	\$2,973	\$1,725	\$1,368	\$1,011	\$879	\$5,000	18%
CPA	\$63	\$47	\$51	\$30	\$46	30	154%
New Leads							

It is clear that an average user can infer what the original results mean from the query, however, it is not in the client's best interest to have to put effort into understanding how and where the information is gathered from. The client's simply need a neat and clear representation of how their business is performing both financially and growth-wise. They only need the most important metrics.

My Project - A Scheduling Algorithm using Data from a Database

Introduction

I noticed that there were a lot of discrepancies in the scheduling system for student-workers. There was a lack of attention to detail when management roles were scheduling students on which created friction as some people were given significantly more hours than others and some shifts did not comply with the availability sent by the students themselves. It would be easier on the management team, more reliable for the students and more time-efficient than manually creating a schedule.

My Solution

I have created an algorithm that reads in student's availability schedule taken from a relational database (in MySQL) and distributes the weekly hours fairly. By creating a connection between MySQL and Python programs, I was able to produce a reliable schedule that should not have any discrepancies and is time-efficient. A computer leaves no room for favoritism and my program should present each individuals total hours for the week. This should also improve the time-sheet approval process on the management end as if everyone shows up to their scheduled shift, there should be no need for extra verification.

Relating to my Work at Solve

The sort of queries I wrote against the database produced in this project resemble those that I would create in the office at Solve. I do not create nor edit the data within databases at work, however, I found this process simpler to understand due to the conceptual grasp I have from both work and my CS306 class. I want to show that regardless of the information within the database, I can apply the skill and knowledge taken from the job to anything.

The Interaction between Python and MySQL

I used a local MySQL server to create a connection between my python program and the database. Using the mysql library, I was able to create a function that would create the connection or return whether it ran into an error.

```
# connect to the database (requires db password)
def create_server_connection(host_name, user_name, user_password):
    connection = None
    try:
        connection = mysql.connector.connect(
            host=host_name,
            user=user_name,
            passwd=user_password
        )
        print("MySQL Database connection successful")
    except Error as err:
        print(f"Error: '{err}'")

    return connection
```

The Database

There is a one to one relationship between the students and student_availability tables. The student table consists of the id, first name, last name, working department and the students class level. The student_availability table includes id, availability which is a json type and week beginning which is a date type.

```
mysql> show tables;
+-----+
| Tables_in-coop_project |
+-----+
| student                 |
| student_availability    |
+-----+
2 rows in set (0.00 sec)
```

The Algorithm

From the database, I queried the student_availability table to return the student ids of those who were available to work the relative shift; if the available start time was less than the start time of the shift and the available end time was larger than the end time of the shift. This information was stored in an array in Python and using the random library, I selected a random ID from the workers available. Using the relational database, I identified who's name coincided with the particular ID chosen to work the shift and created a table to display the schedule for the week.

Specifications

Product Features

The program, given the weekly availability of each student, is able to produce a schedule file that should have no availability discrepancies. This would eliminate problems between students and staff and reduce time spent on scheduling.

Software Requirements and Prerequisites

- System must have MySQL installed on the computer
- prettytable must be installed in the IDE
- The student availability should be formatted in a JSON format using brackets as key : value instances and should be entered as 24-hour time without any formatting as such: 2pm would be 1400

Performance Details

This program must interact with the database as well as interpret and use the data returned. The program created will create a connection between a MySQL local server and the python program. A database will be created, populated with student information and queried. The data will be interpreted and manipulated to produce a schedule according to the availability of the students within the database.

Limitations

- This is a weekly schedule. The program must be run more frequently
- If there is a differing availability of a student per week, the JSON field must be hardcoded in and updated with any change

Program Run-through

Creating a Connection to MySQL and Creating a Database

```
# connect to the database (requires db password)
def create_server_connection(host_name, user_name, user_password):
    connection = None
    try:
        connection = mysql.connector.connect(
            host=host_name,
            user=user_name,
            passwd=user_password
        )
        print("MySQL Database connection successful")
    except Error as err:
        print(f"Error: '{err}'")

    return connection

# create database
def create_database(connection, query):
    cursor = connection.cursor()
    try:
        cursor.execute(query)
        print("Database created successfully")
    except Error as err:
        print(f"Error: '{err}'")
```

The functions to the left are used to create a server connection and database. The `create_server_connection` uses the localhost and root as well as the server's password to create a connection. The error field ensures that if the database returns an error, the IDE will display it too. The `create_database` function creates an empty database in MySQL.

Populating the Database

```
create_database_query = "CREATE DATABASE coop_project"

create_student_table = """
CREATE TABLE student(
    id INT PRIMARY KEY,
    first_name VARCHAR(16) NOT NULL,
    last_name VARCHAR(20) NOT NULL,
    department VARCHAR(20) NOT NULL,
    class VARCHAR(10) NOT NULL
);
"""

create_student_availability_table = """
CREATE TABLE student_availability(
    id INT PRIMARY KEY,
    availability json NOT NULL,
    week_beginning DATE NOT NULL
);
"""

alter_student_availability_fk = """
ALTER TABLE student_availability
ADD FOREIGN KEY (id) REFERENCES student(id)
"""
```

In order to run queries from python to MySQL, the query itself must be stored as a string in the exact syntax that would be run in SQL. Each field in the student table was initialized with its data type and it was ensured that there were no blank entries. A primary key is a unique identifier for the table. The communication between MySQL and python did not allow multiple line read to work accurately when creating a foreign key while initializing the table. As a result I had to run a separate query

(`alter_student_availability`) to ensure that the tables were related by a foreign key (ID).

```

insert_s8 = 'INSERT INTO student(id, first_name, last_name, department, class) VALUES (008, "Hannah", "Alvarado", "Front Desk", "Junior");'

insert_s1_avail = """"INSERT INTO student_availability(id, availability, week_beginning) VALUES(001,
    '{
      "Monday" : {
        "start" : "0600",
        "end" : "1400"},
      "Tuesday" : {
        "start" : "null",
        "end" : "null"},
      "Wednesday": {
        "start" : "0600",
        "end" : "1400" },
      "Thursday" : {
        "start" : "1200",
        "end" : "2200"},
      "Friday" : {
        "start" : "0600",
        "end" : "1800"},
      "Saturday" : {
        "start" : "null",
        "end" : "null"},
      "Sunday" : {
        "start" : "1000",
        "end" : "1600"}
    }',
    '2022-12-05');""""

```

The insert_s8 line of code is one example of populating the student table with relevant student information in each field. The student table requires id, a primary key that must be unique within the table, the first and last name of the student, their class level and the department they're working for. The student_availability table requires id (as a foreign key to relate the two tables), their availability written in JSON format and the date of the monday from the week as week_beginning.

In order to ensure that there would be overlap of student availabilities, I created 8 students with different availabilities and counted where there was overlap by ID as shown in the following table.

Shifts	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
6am - 10am	1, 5, 7	4, 6	1, 6, 8	4, 8	1, 6	-	-
10am - 2pm	1, 2, 5, 7	3, 5, 6, 8	1, 3, 6, 8	3, 8	1, 2, 6	-	-
2pm - 6pm	3, 4, 5	3, 6, 7	3, 5, 7	1, 6, 7, 8	1, 2, 6, 8	-	-
6pm - 10pm	3, 4, 6, 8	2, 3, 7	5, 7	1, 6, 7	3, 7, 8	-	-
7am - 12pm	-	-	-	-	-	2, 4, 6, 8	6, 7
12pm - 5pm	-	-	-	-	-	6	2, 6, 8

Querying the Database

```
m_first_shift_avail = ""
SELECT ID
FROM student_availability
WHERE CAST(JSON_EXTRACT(availability, "$.Monday.start") AS DECIMAL) <= 600
AND CAST(JSON_EXTRACT(availability, "$.Monday.end") AS DECIMAL) >= 1000;
""
```

I hardcoded this same piece of code for each possible shift to tell the database to return the student ids of all students whos

start time was less than the start of the shift and their end availability time was larger than the time of the end of the shift. In this case if their start available time was less than or equal to 0600 (6am) and their end available time is larger than or equal to 1000 (10am).

```
get_names = ""
SELECT id,
       first_name,
       last_name
FROM student
""
```

I also retrieved the ids and full names from the student table.

Using the Data to Create a Schedule

When information is returned from the database, it is not formatted in a python-friendly manner as such. In order to access the usable and relevant information, I created a function that would loop through the results and append the needed

information into a new array. This recognised the variables as integers rather than strings which enabled me

```
# creates arrays from the data returned
def create_list_from_data(list_name, results):
    for id in results:
        # results are returned in a bracketed format, this is eliminated here
        split = [*id]
        list_name.append(split[0])
```

to more easily relate this information to find the names that identify with that id. The result of calling this function is:

```
178 # run through for loop from query of return and store names in array
179 create_list_from_data(m1, m1_array)
180 print(m1)
181
```

OUTPUT	TERMINAL	JUPYTER	DEBUG CONSOLE
<pre>• (base) elizaclamor@Elizas-MacBook-Pro COOP % /Users/elizaclamor/miniconda/bin/p /fall 2022/COOP/project.py" MySQL Database connection successful MySQL Database connection successful [1, 5, 7]</pre>			

Creating the Schedule

Using the random library, I used random.choice and handed that the array of available student-workers to select the id of the

```
79 # picks a worker for the shift using random
80 def pick_shift(available_students):
81     return random.choice(available_students)
```

individual to work the shift. As this is completely random, it is a more fair approach and ensures that bias does not affect the overall schedule.

I created a dictionary that would store the chosen workers id and shift under a specific shift number starting with schedule[0] as the first

```
258 schedule = {}
259
260 schedule[0] = [m1_worker, '6am-10am']
```

monday shift following with schedule[1] as the second monday shift and so on.

```
291 # create dictionary of names and worker ID
292 names_results = read_query(connection, get_names)
293 id_to_name = {}
294 create_student_dict(id_to_name, names_results)
295
296 # change workers to names by for loop going through list
297 for i in range(total_shifts):
298     for id in id_to_name:
299         if schedule[i][0] == id:
300             schedule[i][0] = id_to_name[id]
```

As the names and ids came from another query, I ran through the results and stored them in another dictionary to identify full names from id number. I then looped through all the schedules to change the id

value to their full name.

Finally, I used the prettytable library to create a presentable table for the schedule and wrote that to a file.

This produce the following as the final schedule:

project.py

database.py

print_schedule.py

queries.py

schedulefile.txt

≡ schedulefile.txt

1

Week Beginning : 2022-12-05

2

+

3

Shift Time

Monday

Tuesday

Wednesday

Thursday

Friday

Saturday

Sunday

4

+

5

6am-10am

Nico Teynie

Aaron Mantilla

Aaron Mantilla

Hannah Alvarado

Vanshita Malhotra

-

-

6

10am-2pm

Nico Teynie

Nico Teynie

Giada Zorzan

Sara Zorzan

Vanshita Malhotra

-

-

7

2pm-6pm

Nico Teynie

Priyam Shah

Sara Zorzan

Aaron Mantilla

Michael Cross

-

-

8

6pm-10pm

Giada Zorzan

Sara Zorzan

Nico Teynie

Priyam Shah

Priyam Shah

-

-

9

7am-12pm

-

-

-

-

-

Giada Zorzan

Aaron Mantilla

10

12pm-5pm

-

-

-

-

-

Aaron Mantilla

Aaron Mantilla

11

+

Notes

This algorithm does not account for when there are long stretches of availabilities and someone is scheduled on twice in the same day but there is a split between their shift. An example would be:

```
insert_s5_avail = ""INSERT INTO student_availability(id, availability, week_beginning) VALUES(005,
  '{
    "Monday" : {
      "start" : "0600",
      "end" : "1800"},
    "Tuesday" : {
      "start" : "1000",
      "end" : "1500"},
    "Wednesday" : {
      "start" : "1300",
      "end" : "2200"},
    "Thursday" : {
      "start" : "null",
      "end" : "null"},
    "Friday" : {
      "start" : "null",
      "end" : "null"},
    "Saturday" : {
      "start" : "null",
      "end" : "null"},
    "Sunday" : {
      "start" : "null",
      "end" : "null"}
  }', '2022-12-05');""
```

As we can see, Nico Teynie is scheduled twice within the same day however there is a split between his shift. In the future, I would love to improve this issue.

Student with id number 5 is identified as Nico Teynie. It is clear that his monday availability is very broad. This can cause implications as such:

Week Beginning : 2022-12-05		
Shift Time	Monday	
6am-10am	Nico Teynie	
10am-2pm	Priyam Shah	
2pm-6pm	Nico Teynie	
6pm-10pm	Giada Zorzan	
7am-12pm	-	
12pm-5pm	-	

Code

project.py

```
08/12/2022, 00:03 project.py
1 # project main file
2
3 # imports
4 import mysql.connector
5 from mysql.connector import Error
6 from database import *
7 from queries import *
8 import random
9
10 # connect to the database (requires db password)
11 def create_server_connection(host_name, user_name, user_password):
12     connection = None
13     try:
14         connection = mysql.connector.connect(
15             host=host_name,
16             user=user_name,
17             passwd=user_password
18         )
19         print("MySQL Database connection successful")
20     except Error as err:
21         print(f"Error: '{err}'")
22     return connection
23
24 # create database
25 def create_database(connection, query):
26     cursor = connection.cursor()
27     try:
28         cursor.execute(query)
29         print("Database created successfully")
30     except Error as err:
31         print(f"Error: '{err}'")
32
33 # create connection
34 def create_db_connection(host_name, user_name, user_password, db_name):
35     connection = None
36     try:
37         connection = mysql.connector.connect(
38             host=host_name,
39             user=user_name,
40             passwd=user_password,
41             database=db_name
42         )
43         print("MySQL Database connection successful")
44     except Error as err:
45         print(f"Error: '{err}'")
46     return connection
47
48 # run queries
49 def execute_query(connection, query, multi):
50     cursor = connection.cursor()
51     try:
52         cursor.execute(query, multi)
53         connection.commit()
54         print("Query successful")
55     except Error as err:
56         print(f"Error: '{err}'")
57
58 localhost:4649~/moderspython 1/6
```

```
08/12/2022, 00:03 project.py
60
61 # read queries
62 def read_query(connection, query):
63     cursor = connection.cursor()
64     result = None
65     try:
66         cursor.execute(query)
67         result = cursor.fetchall()
68         return result
69     except Error as err:
70         print(f"Error: '{err}'")
71
72 # creates arrays from the data returned
73 def create_list_from_data(list_name, results):
74     for id in results:
75         # results are returned in a bracketed format, this is eliminated here
76         split = [*id]
77         list_name.append(split[0])
78
79 # picks a worker for the shift using random
80 def pick_shift(available_students):
81     return random.choice(available_students)
82
83 # create a dictionary to identify the names from the student worker id
84 def create_student_dict(dict_name, results):
85     # identify key as id and value as full name
86     for item in results:
87         name = item[1] + " " + item[2]
88         dict_name[item[0]] = name
89     return dict_name
90
91
92
93 # Can move this section later to a different file for tidiness
94 # testing validity
95 pw = "Youwillbeokay!"
96 connection = create_server_connection("localhost", "root", pw)
97
98 # create database
99 # create_database(connection, create_database_query)
100 db = 'coop_project'
101 connection = create_db_connection("localhost", "root", pw, db)
102
103 # create tables
104 execute_query(connection, create_student_table, False)
105 execute_query(connection, create_student_availability_table, False)
106 execute_query(connection, alter_student_availability_fk, False)
107
108 # populate student table
109 execute_query(connection, insert_s1, False)
110 execute_query(connection, insert_s2, False)
111 execute_query(connection, insert_s3, False)
112 execute_query(connection, insert_s4, False)
113 execute_query(connection, insert_s5, False)
114 execute_query(connection, insert_s6, False)
115 execute_query(connection, insert_s7, False)
116 execute_query(connection, insert_s8, False)
117
118 # populate student availability table
119 execute_query(connection, insert_s1_avail, False)
120
121 localhost:4649~/moderspython 2/6
```

```
08/12/2022, 00:03 project.py
120 execute_query(connection, insert_s2_avail, False)
121 execute_query(connection, insert_s3_avail, False)
122 execute_query(connection, insert_s4_avail, False)
123 execute_query(connection, insert_s5_avail, False)
124 execute_query(connection, insert_s6_avail, False)
125 execute_query(connection, insert_s7_avail, False)
126 execute_query(connection, insert_s8_avail, False)
127
128 # retrieving those who are available to work the shift
129
130 # monday shifts
131 m1_array = read_query(connection, m_first_shift_avail)
132 m2_array = read_query(connection, m_second_shift_avail)
133 m3_array = read_query(connection, m_third_shift_avail)
134 m4_array = read_query(connection, m_fourth_shift_avail)
135
136 # tuesday shifts
137 t1_array = read_query(connection, t_first_shift_avail)
138 t2_array = read_query(connection, t_second_shift_avail)
139 t3_array = read_query(connection, t_third_shift_avail)
140 t4_array = read_query(connection, t_fourth_shift_avail)
141
142 # wednesday shifts
143 w1_array = read_query(connection, w_first_shift_avail)
144 w2_array = read_query(connection, w_second_shift_avail)
145 w3_array = read_query(connection, w_third_shift_avail)
146 w4_array = read_query(connection, w_fourth_shift_avail)
147
148 # thursday shifts
149 th1_array = read_query(connection, th_first_shift_avail)
150 th2_array = read_query(connection, th_second_shift_avail)
151 th3_array = read_query(connection, th_third_shift_avail)
152 th4_array = read_query(connection, th_fourth_shift_avail)
153
154 # friday shifts
155 f1_array = read_query(connection, f_first_shift_avail)
156 f2_array = read_query(connection, f_second_shift_avail)
157 f3_array = read_query(connection, f_third_shift_avail)
158 f4_array = read_query(connection, f_fourth_shift_avail)
159
160 # saturday shifts
161 s1_array = read_query(connection, s_first_shift_avail)
162 s2_array = read_query(connection, s_second_shift_avail)
163
164 # sunday shifts
165 su1_array = read_query(connection, su_first_shift_avail)
166 su2_array = read_query(connection, su_second_shift_avail)
167
168
169 # empty lists to store available ids
170 m1, m2, m3, m4 = [], [], [], []
171 t1, t2, t3, t4 = [], [], [], []
172 w1, w2, w3, w4 = [], [], [], []
173 th1, th2, th3, th4 = [], [], [], []
174 f1, f2, f3, f4 = [], [], [], []
175 s1, s2 = [], []
176 su1, su2 = [], []
177
178 # run through for loop from query of return and store names in array
179 create_list_from_data(m1, m1_array)
180
181 localhost:4649~/moderspython 3/6
```

```
08/12/2022, 00:03 project.py
180 create_list_from_data(m2, m2_array)
181 create_list_from_data(m3, m3_array)
182 create_list_from_data(m4, m4_array)
183
184 create_list_from_data(t1, t1_array)
185 create_list_from_data(t2, t2_array)
186 create_list_from_data(t3, t3_array)
187 create_list_from_data(t4, t4_array)
188
189 create_list_from_data(w1, w1_array)
190 create_list_from_data(w2, w2_array)
191 create_list_from_data(w3, w3_array)
192 create_list_from_data(w4, w4_array)
193
194 create_list_from_data(th1, th1_array)
195 create_list_from_data(th2, th2_array)
196 create_list_from_data(th3, th3_array)
197 create_list_from_data(th4, th4_array)
198
199 create_list_from_data(f1, f1_array)
200 create_list_from_data(f2, f2_array)
201 create_list_from_data(f3, f3_array)
202 create_list_from_data(f4, f4_array)
203
204 create_list_from_data(s1, s1_array)
205 create_list_from_data(s2, s2_array)
206
207 create_list_from_data(su1, su1_array)
208 create_list_from_data(su2, su2_array)
209
210
211 # algorithm to randomly choose individual to work
212 # use rand to select a name
213
214 # shifts totals
215 week = 7
216 weekday_shifts = 4
217 weekend_shifts = 2
218
219 total_shifts = (5 * weekday_shifts) + (2 * weekend_shifts)
220
221 # picking workers for shifts
222 m1_worker = pick_shift(m1)
223 m2_worker = pick_shift(m2)
224 m3_worker = pick_shift(m3)
225 m4_worker = pick_shift(m4)
226
227 t1_worker = pick_shift(t1)
228 t2_worker = pick_shift(t2)
229 t3_worker = pick_shift(t3)
230 t4_worker = pick_shift(t4)
231
232 w1_worker = pick_shift(w1)
233 w2_worker = pick_shift(w2)
234 w3_worker = pick_shift(w3)
235 w4_worker = pick_shift(w4)
236
237 th1_worker = pick_shift(th1)
238 th2_worker = pick_shift(th2)
239 th3_worker = pick_shift(th3)
240
241 localhost:4649~/moderspython 4/6
```

```
08/12/2022, 00:03 project.py
240 th4_worker = pick_shift(th4)
241
242 f1_worker = pick_shift(f1)
243 f2_worker = pick_shift(f2)
244 f3_worker = pick_shift(f3)
245 f4_worker = pick_shift(f4)
246
247 s1_worker = pick_shift(s1)
248 s2_worker = pick_shift(s2)
249
250 su1_worker = pick_shift(s1)
251 su2_worker = pick_shift(s2)
252
253 # once schedule is generated, go through schedule and check if individual is
working twice in the same day
254 # if they are not working the day already or are working the shift before or
after.
255 schedule = {}
256
257 schedule[0] = [m1_worker, '6am-10am']
258 schedule[1] = [m2_worker, '10am-2pm']
259 schedule[2] = [m3_worker, '2pm-6pm']
260 schedule[3] = [m4_worker, '6pm-10pm']
261
262 schedule[4] = [t1_worker, '6am-10am']
263 schedule[5] = [t2_worker, '10am-2pm']
264 schedule[6] = [t3_worker, '2pm-6pm']
265 schedule[7] = [t4_worker, '6pm-10pm']
266
267 schedule[8] = [w1_worker, '6am-10am']
268 schedule[9] = [w2_worker, '10am-2pm']
269 schedule[10] = [w3_worker, '2pm-6pm']
270 schedule[11] = [w4_worker, '6pm-10pm']
271
272 schedule[12] = [th1_worker, '6am-10am']
273 schedule[13] = [th2_worker, '10am-2pm']
274 schedule[14] = [th3_worker, '2pm-6pm']
275 schedule[15] = [th4_worker, '6pm-10pm']
276
277 schedule[16] = [f1_worker, '6am-10am']
278 schedule[17] = [f2_worker, '10am-2pm']
279 schedule[18] = [f3_worker, '2pm-6pm']
280 schedule[19] = [f4_worker, '6pm-10pm']
281
282 schedule[20] = [s1_worker, '7am-12pm']
283 schedule[21] = [s2_worker, '12pm-5pm']
284
285 schedule[22] = [su1_worker, '7am-12pm']
286 schedule[23] = [su2_worker, '12pm-5pm']
287
288 # create dictionary of names and worker ID
289 names_results = read_query(connection, get_names)
290 id_to_name = {}
291 create_student_dict(id_to_name, names_results)
292
293 # change workers to names by for loop going through list
294 for i in range(total_shifts):
295     for id in id_to_name:
296         if schedule[i][0] == id:
297             schedule[i][0] = id_to_name[id]

```

localhost:4649/mode=python 5/6

```
08/12/2022, 00:03 project.py
298
299 # create file that shows the schedule
300 # in print schedule file
301
302 # get date for the schedule
303 date = read_query(connection, get_week)
304

```

localhost:4649/mode=python 6/6

database.py

```
08/12/2022, 00:07 database.py
1 ''' Database set up for project
2 include tables: STUDENT, STUDENT_AVAILABILITY'''
3
4 create_database_query = "CREATE DATABASE coop_project"
5
6 create_student_table = """
7 CREATE TABLE student(
8     id INT PRIMARY KEY,
9     first_name VARCHAR(16) NOT NULL,
10    last_name VARCHAR(20) NOT NULL,
11    department VARCHAR(20) NOT NULL,
12    class VARCHAR(10) NOT NULL
13 );
14 """
15
16 create_student_availability_table = """
17 CREATE TABLE student_availability(
18     id INT PRIMARY KEY,
19     availability json NOT NULL,
20     week_beginning DATE NOT NULL
21 );
22 """
23
24 alter_student_availability_fk = """
25 ALTER TABLE student_availability
26 ADD FOREIGN KEY (id) REFERENCES student(id)
27 """
28
29 insert_s1 = 'INSERT INTO student(id, first_name, last_name, department,
class) VALUES (001, "Vanshita", "Malhotra", "Front Desk", "Sophomore");'
30 insert_s2 = 'INSERT INTO student(id, first_name, last_name, department,
class) VALUES (002, "Michael", "Cross", "Front Desk", "Sophomore");'
31 insert_s3 = 'INSERT INTO student(id, first_name, last_name, department,
class) VALUES (003, "Sara", "Zorzan", "Front Desk", "Sophomore");'
32 insert_s4 = 'INSERT INTO student(id, first_name, last_name, department,
class) VALUES (004, "Giada", "Zorzan", "Front Desk", "Junior");'
33 insert_s5 = 'INSERT INTO student(id, first_name, last_name, department,
class) VALUES (005, "Nico", "Teyndie", "Front Desk", "Senior");'
34 insert_s6 = 'INSERT INTO student(id, first_name, last_name, department,
class) VALUES (006, "Aaron", "Mantilla", "Front Desk", "Senior");'
35 insert_s7 = 'INSERT INTO student(id, first_name, last_name, department,
class) VALUES (007, "Priyam", "Shah", "Front Desk", "Graduate");'
36 insert_s8 = 'INSERT INTO student(id, first_name, last_name, department,
class) VALUES (008, "Hannah", "Alvarado", "Front Desk", "Junior");'
37
38 insert_s1_avail = """INSERT INTO student_availability(id, availability,
week_beginning) VALUES(001,
39 {
40     "Monday" : {
41         "start" : "0600",
42         "end" : "1400"},
43     "Tuesday" : {
44         "start" : "null",
45         "end" : "null"},
46     "Wednesday" : {
47         "start" : "0600",
48         "end" : "1400"},
49     "Thursday" : {
50         "start" : "1200",

```

localhost:4649/mode=python 1/5

```
08/12/2022, 00:07 database.py
51     "end" : "2200"},
52 "Friday" : {
53     "start" : "0600",
54     "end" : "1800"},
55 "Saturday" : {
56     "start" : "null",
57     "end" : "null"},
58 "Sunday" : {
59     "start" : "1000",
60     "end" : "1600"}
61 },
62 '2022-12-05');"""
63 insert_s2_avail = """INSERT INTO student_availability(id, availability,
week_beginning) VALUES(002,
64 {
65     "Monday" : {
66         "start" : "0800",
67         "end" : "1300"},
68     "Tuesday" : {
69         "start" : "1500",
70         "end" : "2200"},
71     "Wednesday" : {
72         "start" : "1500",
73         "end" : "1900"},
74     "Thursday" : {
75         "start" : "null",
76         "end" : "null"},
77     "Friday" : {
78         "start" : "0800",
79         "end" : "1800"},
80     "Saturday" : {
81         "start" : "0700",
82         "end" : "1500"},
83     "Sunday" : {
84         "start" : "1200",
85         "end" : "2000"}
86 },
87 '2022-12-05');"""
88 insert_s3_avail = """INSERT INTO student_availability(id, availability,
week_beginning) VALUES(003,
89 {
90     "Monday" : {
91         "start" : "null",
92         "end" : "null"},
93     "Tuesday" : {
94         "start" : "0900",
95         "end" : "2200"},
96     "Wednesday" : {
97         "start" : "1000",
98         "end" : "2000"},
99     "Thursday" : {
100        "start" : "0800",
101        "end" : "1400"},
102    "Friday" : {
103        "start" : "1500",
104        "end" : "2300"},
105    "Saturday" : {
106        "start" : "0600",
107        "end" : "1300"},
108    "Sunday" : {
109        "start" : "0600",

```

localhost:4649/mode=python 2/5

```

08/12/2022, 00:07 database.py
109         "end" : "1200"}
110     }, '2022-12-05');"""
111 insert_s4_avail = """INSERT INTO student_availability(id, availability,
week_beginning) VALUES(004,
112     '{"Monday" : {
113         "start" : "1300",
114         "end" : "2200"},
115     "Tuesday" : {
116         "start" : "0600",
117         "end" : "1200"},
118     "Wednesday": {
119         "start" : "1000",
120         "end" : "1700"},
121     "Thursday" : {
122         "start" : "0500",
123         "end" : "1300"},
124     "Friday" : {
125         "start" : "null",
126         "end" : "null"},
127     "Saturday" : {
128         "start" : "0700",
129         "end" : "1300"},
130     "Sunday" : {
131         "start" : "null",
132         "end" : "null"}
133     }', '2022-12-05');"""
134 insert_s5_avail = """INSERT INTO student_availability(id, availability,
week_beginning) VALUES(005,
135     '{"Monday" : {
136         "start" : "0600",
137         "end" : "1800"},
138     "Tuesday" : {
139         "start" : "1000",
140         "end" : "1500"},
141     "Wednesday": {
142         "start" : "1300",
143         "end" : "2200"},
144     "Thursday" : {
145         "start" : "null",
146         "end" : "null"},
147     "Friday" : {
148         "start" : "null",
149         "end" : "null"},
150     "Saturday" : {
151         "start" : "null",
152         "end" : "null"},
153     "Sunday" : {
154         "start" : "null",
155         "end" : "null"}
156     }', '2022-12-05');"""
157 insert_s6_avail = """INSERT INTO student_availability(id, availability,
week_beginning) VALUES(006,
158     '{"Monday" : {
159         "start" : "1700",
160         "end" : "2200"},
161     "Tuesday" : {
162         "start" : "0600",
163         "end" : "1800"},
164     "Wednesday": {
165         "start" : "0600",

```

localhost:4649/?mode=python

3/5

```

08/12/2022, 00:07 database.py
166         "end" : "1400"},
167     "Thursday" : {
168         "start" : "1300",
169         "end" : "2200"},
170     "Friday" : {
171         "start" : "0600",
172         "end" : "1800"},
173     "Saturday" : {
174         "start" : "0700",
175         "end" : "1700"},
176     "Sunday" : {
177         "start" : "0700",
178         "end" : "1700"}
179     }', '2022-12-05');"""
180 insert_s7_avail = """INSERT INTO student_availability(id, availability,
week_beginning) VALUES(007,
181     '{"Monday" : {
182         "start" : "0600",
183         "end" : "1500"},
184     "Tuesday" : {
185         "start" : "1300",
186         "end" : "2300"},
187     "Wednesday": {
188         "start" : "1300",
189         "end" : "2300"},
190     "Thursday" : {
191         "start" : "1300",
192         "end" : "2300"},
193     "Friday" : {
194         "start" : "1700",
195         "end" : "2300"},
196     "Saturday" : {
197         "start" : "null",
198         "end" : "null"},
199     "Sunday" : {
200         "start" : "0700",
201         "end" : "1200"}
202     }', '2022-12-05');"""
203 insert_s8_avail = """INSERT INTO student_availability(id, availability,
week_beginning) VALUES(008,
204     '{"Monday" : {
205         "start" : "1800",
206         "end" : "2300"},
207     "Tuesday" : {
208         "start" : "1000",
209         "end" : "1400"},
210     "Wednesday": {
211         "start" : "0600",
212         "end" : "1400"},
213     "Thursday" : {
214         "start" : "0600",
215         "end" : "1800"},
216     "Friday" : {
217         "start" : "1300",
218         "end" : "2200"},
219     "Saturday" : {
220         "start" : "0700",
221         "end" : "1200"},
222     "Sunday" : {
223         "start" : "1200",

```

localhost:4649/?mode=python

4/5

```

08/12/2022, 00:07 database.py
224         "end" : "1700"}
225     }, '2022-12-05');"""
226
227

```

localhost:4649/?mode=python

5/5

queries.py

```
08/12/2022, 00:08 queries.py
1 ''' Queries for the project
2
3 SHIFTS
4 MTWThF: 6-10, 10-2, 2-6, 6-10
5 SSu: 7-12, 12-5
6 '''
7
8 # these shifts are hard coded
9 # returns id that is available for the shift
10 m_first_shift_avail = ""
11 SELECT ID
12 FROM student_availability
13 WHERE CAST(JSON_EXTRACT(availability, "$.Monday.start") AS DECIMAL) <= 600
14 AND CAST(JSON_EXTRACT(availability, "$.Monday.end") AS DECIMAL) >= 1000;
15 ""
16 t_first_shift_avail = ""
17 SELECT ID
18 FROM student_availability
19 WHERE CAST(JSON_EXTRACT(availability, "$.Tuesday.start") AS DECIMAL) <= 600
20 AND CAST(JSON_EXTRACT(availability, "$.Tuesday.end") AS DECIMAL) >= 1000;
21 ""
22 w_first_shift_avail = ""
23 SELECT ID
24 FROM student_availability
25 WHERE CAST(JSON_EXTRACT(availability, "$.Wednesday.start") AS DECIMAL) <= 600
26 AND CAST(JSON_EXTRACT(availability, "$.Wednesday.end") AS DECIMAL) >= 1000;
27 ""
28 th_first_shift_avail = ""
29 SELECT ID
30 FROM student_availability
31 WHERE CAST(JSON_EXTRACT(availability, "$.Thursday.start") AS DECIMAL) <= 600
32 AND CAST(JSON_EXTRACT(availability, "$.Thursday.end") AS DECIMAL) >= 1000;
33 ""
34 f_first_shift_avail = ""
35 SELECT ID
36 FROM student_availability
37 WHERE CAST(JSON_EXTRACT(availability, "$.Friday.start") AS DECIMAL) <= 600
38 AND CAST(JSON_EXTRACT(availability, "$.Friday.end") AS DECIMAL) >= 1000;
39 ""
40 s_first_shift_avail = ""
41 SELECT ID
42 FROM student_availability
43 WHERE CAST(JSON_EXTRACT(availability, "$.Saturday.start") AS DECIMAL) <= 700
44 AND CAST(JSON_EXTRACT(availability, "$.Saturday.end") AS DECIMAL) >= 1200;
45 ""
46 su_first_shift_avail = ""
47 SELECT ID
48 FROM student_availability
49 WHERE CAST(JSON_EXTRACT(availability, "$.Sunday.start") AS DECIMAL) <= 700
50 AND CAST(JSON_EXTRACT(availability, "$.Sunday.end") AS DECIMAL) >= 1200;
51 ""
52
53 m_second_shift_avail = ""
54 SELECT ID
55 FROM student_availability
56 WHERE CAST(JSON_EXTRACT(availability, "$.Monday.start") AS DECIMAL) <= 1000
57 AND CAST(JSON_EXTRACT(availability, "$.Monday.end") AS DECIMAL) >= 1400;
58 ""
59 t_second_shift_avail = ""
```

localhost:4649?mode=python

1/3

```
08/12/2022, 00:08 queries.py
60 SELECT ID
61 FROM student_availability
62 WHERE CAST(JSON_EXTRACT(availability, "$.Tuesday.start") AS DECIMAL) <= 1000
63 AND CAST(JSON_EXTRACT(availability, "$.Tuesday.end") AS DECIMAL) >= 1400;
64 ""
65 w_second_shift_avail = ""
66 SELECT ID
67 FROM student_availability
68 WHERE CAST(JSON_EXTRACT(availability, "$.Wednesday.start") AS DECIMAL) <=
1000
69 AND CAST(JSON_EXTRACT(availability, "$.Wednesday.end") AS DECIMAL) >= 1400;
70 ""
71 th_second_shift_avail = ""
72 SELECT ID
73 FROM student_availability
74 WHERE CAST(JSON_EXTRACT(availability, "$.Thursday.start") AS DECIMAL) <= 1000
75 AND CAST(JSON_EXTRACT(availability, "$.Thursday.end") AS DECIMAL) >= 1400;
76 ""
77 f_second_shift_avail = ""
78 SELECT ID
79 FROM student_availability
80 WHERE CAST(JSON_EXTRACT(availability, "$.Friday.start") AS DECIMAL) <= 1000
81 AND CAST(JSON_EXTRACT(availability, "$.Friday.end") AS DECIMAL) >= 1400;
82 ""
83 s_second_shift_avail = ""
84 SELECT ID
85 FROM student_availability
86 WHERE CAST(JSON_EXTRACT(availability, "$.Saturday.start") AS DECIMAL) <= 1200
87 AND CAST(JSON_EXTRACT(availability, "$.Saturday.end") AS DECIMAL) >= 1700;
88 ""
89 su_second_shift_avail = ""
90 SELECT ID
91 FROM student_availability
92 WHERE CAST(JSON_EXTRACT(availability, "$.Sunday.start") AS DECIMAL) <= 1200
93 AND CAST(JSON_EXTRACT(availability, "$.Sunday.end") AS DECIMAL) >= 1700;
94 ""
95
96 m_third_shift_avail = ""
97 SELECT ID
98 FROM student_availability
99 WHERE CAST(JSON_EXTRACT(availability, "$.Monday.start") AS DECIMAL) <= 1400
100 AND CAST(JSON_EXTRACT(availability, "$.Monday.end") AS DECIMAL) >= 1800;
101 ""
102 t_third_shift_avail = ""
103 SELECT ID
104 FROM student_availability
105 WHERE CAST(JSON_EXTRACT(availability, "$.Tuesday.start") AS DECIMAL) <= 1400
106 AND CAST(JSON_EXTRACT(availability, "$.Tuesday.end") AS DECIMAL) >= 1800;
107 ""
108 w_third_shift_avail = ""
109 SELECT ID
110 FROM student_availability
111 WHERE CAST(JSON_EXTRACT(availability, "$.Wednesday.start") AS DECIMAL) <=
1400
112 AND CAST(JSON_EXTRACT(availability, "$.Wednesday.end") AS DECIMAL) >= 1800;
113 ""
114 th_third_shift_avail = ""
115 SELECT ID
116 FROM student_availability
117 WHERE CAST(JSON_EXTRACT(availability, "$.Thursday.start") AS DECIMAL) <= 1400
```

localhost:4649?mode=python

2/3

```
08/12/2022, 00:08 queries.py
118 AND CAST(JSON_EXTRACT(availability, "$.Thursday.end") AS DECIMAL) >= 1800;
119 ""
120 f_third_shift_avail = ""
121 SELECT ID
122 FROM student_availability
123 WHERE CAST(JSON_EXTRACT(availability, "$.Friday.start") AS DECIMAL) <= 1400
124 AND CAST(JSON_EXTRACT(availability, "$.Friday.end") AS DECIMAL) >= 1800;
125 ""
126
127 m_fourth_shift_avail = ""
128 SELECT ID
129 FROM student_availability
130 WHERE CAST(JSON_EXTRACT(availability, "$.Monday.start") AS DECIMAL) <= 1800
131 AND CAST(JSON_EXTRACT(availability, "$.Monday.end") AS DECIMAL) >= 2200;
132 ""
133 t_fourth_shift_avail = ""
134 SELECT ID
135 FROM student_availability
136 WHERE CAST(JSON_EXTRACT(availability, "$.Tuesday.start") AS DECIMAL) <= 1800
137 AND CAST(JSON_EXTRACT(availability, "$.Tuesday.end") AS DECIMAL) >= 2200;
138 ""
139 w_fourth_shift_avail = ""
140 SELECT ID
141 FROM student_availability
142 WHERE CAST(JSON_EXTRACT(availability, "$.Wednesday.start") AS DECIMAL) <=
1800
143 AND CAST(JSON_EXTRACT(availability, "$.Wednesday.end") AS DECIMAL) >= 2200;
144 ""
145 th_fourth_shift_avail = ""
146 SELECT ID
147 FROM student_availability
148 WHERE CAST(JSON_EXTRACT(availability, "$.Thursday.start") AS DECIMAL) <= 1800
149 AND CAST(JSON_EXTRACT(availability, "$.Thursday.end") AS DECIMAL) >= 2200;
150 ""
151 f_fourth_shift_avail = ""
152 SELECT ID
153 FROM student_availability
154 WHERE CAST(JSON_EXTRACT(availability, "$.Friday.start") AS DECIMAL) <= 1800
155 AND CAST(JSON_EXTRACT(availability, "$.Friday.end") AS DECIMAL) >= 2200;
156 ""
157
158 get_names = ""
159 SELECT id,
160 first_name,
161 last_name
162 FROM student
163 ""
164
165 get_week = ""
166 SELECT DATE_FORMAT(week_beginning, '%Y-%m-%d') as date
167 FROM student_availability
168 GROUP BY 1;
169 ""
170
```

localhost:4649?mode=python

3/3

print_schedule.py

```
08/12/2022,00:08 print_schedule.py
1 """
2 Function to create file for schedule in easily readable format
3 """
4
5 # imports
6 from project import *
7 from prettytable import PrettyTable
8
9 schedule_file = open("schedulefile.txt", "w")
10 schedule_file.write("Week Beginning : {} \n".format(date[0][0]))
11
12 # creating pretty table
13 schedule_table = PrettyTable(["Shift Time", "Monday", "Tuesday", "Wednesday",
14 "Thursday", "Friday", "Saturday", "Sunday"])
15 schedule_table.add_row([schedule[0][1], schedule[0][0], schedule[4][0],
16 schedule[8][0], schedule[12][0], schedule[16][0], "-", "-"])
17 schedule_table.add_row([schedule[1][1], schedule[1][0], schedule[5][0],
18 schedule[9][0], schedule[13][0], schedule[17][0], "-", "-"])
19 schedule_table.add_row([schedule[2][1], schedule[2][0], schedule[6][0],
20 schedule[10][0], schedule[14][0], schedule[18][0], "-", "-"])
21 schedule_table.add_row([schedule[3][1], schedule[3][0], schedule[7][0],
22 schedule[11][0], schedule[15][0], schedule[19][0], "-", "-"])
23 schedule_table.add_row([schedule[20][1], "-", "-", "-", "-", "-", schedule[20][0],
24 schedule[22][0]])
25 schedule_table.add_row([schedule[21][1], "-", "-", "-", "-", "-", schedule[21][0],
26 schedule[23][0]])
27
28 # writing to the file
29 schedule_file.write(str(schedule_table))
30
31 schedule_file.close()
```

localhost:4649/?mode=python

1/1