SOLVE IO Co-operative Education Project

Fall 2022

Eliza Clamor

Supervisor: Professor Jean-Claude Ngatchou

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 it's services to e-commerce businesses in many different countries such as New Zealand, Australia, and Canada. The key seller about the company is it's 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 nin the product team. We use the Presto SQL query engineer. 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	Trend
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 favoritsm 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.

```python
# 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

```python
# 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

```python
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

```
306    print(m1_array)
307
```

```
OUTPUT   TERMINAL   JUPYTER   DEBUG CONSOLE

(base) elizaclamor@Elizas-MacBook-Pro COOP %
/fall 2022/COOP/project.py"
MySQL Database connection successful
MySQL Database connection successful
[(1,), (5,), (7,)]
```

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

```
# 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])
```

```
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

(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]
```

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 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.

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

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

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

```
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

```
Week Beginning : 2022-12-05
+------------+--------------+----------------+----------------+-----------------+------------------+----------------+----------------+
| Shift Time |    Monday    |    Tuesday     |   Wednesday    |     Thursday    |      Friday      |    Saturday    |     Sunday     |
+------------+--------------+----------------+----------------+-----------------+------------------+----------------+----------------+
|  6am-10am  | Nico Teynie  | Aaron Mantilla | Aaron Mantilla | Hannah Alvarado | Vanshita Malhotra |       -        |       -        |
|  10am-2pm  | Nico Teynie  |  Nico Teynie   |  Giada Zorzan  |   Sara Zorzan   | Vanshita Malhotra |       -        |       -        |
|  2pm-6pm   | Nico Teynie  |  Priyam Shah   |  Sara Zorzan   |  Aaron Mantilla |   Michael Cross   |       -        |       -        |
|  6pm-10pm  | Giada Zorzan |  Sara Zorzan   |  Nico Teynie   |   Priyam Shah   |    Priyam Shah    |       -        |       -        |
|  7am-12pm  |      -       |       -        |       -        |        -        |         -         |  Giada Zorzan  | Aaron Mantilla |
|  12pm-5pm  |      -       |       -        |       -        |        -        |         -         | Aaron Mantilla | Aaron Mantilla |
+------------+--------------+----------------+----------------+-----------------+------------------+----------------+----------------+
```

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


```python
# project main file

# imports
import mysql.connector
from mysql.connector import Error
from database import *
from queries import *
import random

# 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}'")

# create connection
def create_db_connection(host_name, user_name, user_password, db_name):
    connection = None
    try:
        connection = mysql.connector.connect(
            host=host_name,
            user=user_name,
            passwd=user_password,
            database=db_name
        )
        print("MySQL Database connection successful")
    except Error as err:
        print(f"Error: '{err}'")

    return connection

# run queries
def execute_query(connection, query, multitf):
    cursor = connection.cursor()
    try:
        cursor.execute(query, multi=multitf)
        connection.commit()
        print("Query successful")
    except Error as err:
        print(f"Error: '{err}'")

```



```python

# read queries
def read_query(connection, query):
    cursor = connection.cursor()
    result = None
    try:
        cursor.execute(query)
        result = cursor.fetchall()
        return result
    except Error as err:
        print(f"Error: '{err}'")

# 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])

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

# create a dictionary to identify the names from the student worker id
def create_student_dict(dict_name, results):
    # identify key as id and value as full name
    for item in results:
        name = item[1] + " " + item[2]
        dict_name[item[0]] = name

    return dict_name


# Can move this section later to a different file for tidiness
# testing validity
pw = "Youwillbe0kay!"
connection = create_server_connection("localhost", "root", pw)

# create database
# create_database(connection, create_database_query)
db = 'coop_project'
connection = create_db_connection("localhost", "root", pw, db)

# create tables
execute_query(connection, create_student_table, False)
execute_query(connection, create_student_availability_table, False)
execute_query(connection, alter_student_availability_fk, False)

# populate student table
execute_query(connection, insert_s1, False)
execute_query(connection, insert_s2, False)
execute_query(connection, insert_s3, False)
execute_query(connection, insert_s4, False)
execute_query(connection, insert_s5, False)
execute_query(connection, insert_s6, False)
execute_query(connection, insert_s7, False)
execute_query(connection, insert_s8, False)

# populate student availability table
execute_query(connection, insert_s1_avail, False)
```



```python
execute_query(connection, insert_s2_avail, False)
execute_query(connection, insert_s3_avail, False)
execute_query(connection, insert_s4_avail, False)
execute_query(connection, insert_s5_avail, False)
execute_query(connection, insert_s6_avail, False)
execute_query(connection, insert_s7_avail, False)
execute_query(connection, insert_s8_avail, False)

# retreiving those who are available to work the shift

# monday shifts
m1_array = read_query(connection, m_first_shift_avail)
m2_array = read_query(connection, m_second_shift_avail)
m3_array = read_query(connection, m_third_shift_avail)
m4_array = read_query(connection, m_fourth_shift_avail)

# tuesday shifts
t1_array = read_query(connection, t_first_shift_avail)
t2_array = read_query(connection, t_second_shift_avail)
t3_array = read_query(connection, t_third_shift_avail)
t4_array = read_query(connection, t_fourth_shift_avail)

# wednesday shifts
w1_array = read_query(connection, w_first_shift_avail)
w2_array = read_query(connection, w_second_shift_avail)
w3_array = read_query(connection, w_third_shift_avail)
w4_array = read_query(connection, w_fourth_shift_avail)

# thursday shifts
th1_array = read_query(connection, th_first_shift_avail)
th2_array = read_query(connection, th_second_shift_avail)
th3_array = read_query(connection, th_third_shift_avail)
th4_array = read_query(connection, th_fourth_shift_avail)

# friday shifts
f1_array = read_query(connection, f_first_shift_avail)
f2_array = read_query(connection, f_second_shift_avail)
f3_array = read_query(connection, f_third_shift_avail)
f4_array = read_query(connection, f_fourth_shift_avail)

# saturday shifts
s1_array = read_query(connection, s_first_shift_avail)
s2_array = read_query(connection, s_second_shift_avail)

# sunday shifts
su1_array = read_query(connection, su_first_shift_avail)
su2_array = read_query(connection, su_second_shift_avail)


# empty lists to store available ids
m1, m2, m3, m4 = [],[],[],[]
t1, t2, t3, t4 = [],[],[],[]
w1, w2, w3, w4 = [],[],[],[]
th1, th2, th3, th4 = [],[],[],[]
f1, f2, f3, f4 = [],[],[],[]
s1, s2= [],[]
su1, su2 = [],[]

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



```python
create_list_from_data(m2, m2_array)
create_list_from_data(m3, m3_array)
create_list_from_data(m4, m4_array)

create_list_from_data(t1, t1_array)
create_list_from_data(t2, t2_array)
create_list_from_data(t3, t3_array)
create_list_from_data(t4, t4_array)

create_list_from_data(w1, w1_array)
create_list_from_data(w2, w2_array)
create_list_from_data(w3, w3_array)
create_list_from_data(w4, w4_array)

create_list_from_data(th1, th1_array)
create_list_from_data(th2, th2_array)
create_list_from_data(th3, th3_array)
create_list_from_data(th4, th4_array)

create_list_from_data(f1, f1_array)
create_list_from_data(f2, f2_array)
create_list_from_data(f3, f3_array)
create_list_from_data(f4, f4_array)

create_list_from_data(s1, s1_array)
create_list_from_data(s2, s2_array)

create_list_from_data(su1, su1_array)
create_list_from_data(su2, su2_array)


# algorithm to randomly choose individual to work
# use rand to select a name

# shifts totals
week = 7
weekday_shifts = 4
weekend_shifts = 2

total_shifts = (5 * weekday_shifts) + (2 * weekend_shifts)

# picking workers for shifts
m1_worker = pick_shift(m1)
m2_worker = pick_shift(m2)
m3_worker = pick_shift(m3)
m4_worker = pick_shift(m4)

t1_worker = pick_shift(t1)
t2_worker = pick_shift(t2)
t3_worker = pick_shift(t3)
t4_worker = pick_shift(t4)

w1_worker = pick_shift(w1)
w2_worker = pick_shift(w2)
w3_worker = pick_shift(w3)
w4_worker = pick_shift(w4)

th1_worker = pick_shift(th1)
th2_worker = pick_shift(th2)
th3_worker = pick_shift(th3)
```

```
th4_worker = pick_shift(th4)

f1_worker = pick_shift(f1)
f2_worker = pick_shift(f2)
f3_worker = pick_shift(f3)
f4_worker = pick_shift(f4)

s1_worker = pick_shift(s1)
s2_worker = pick_shift(s2)

su1_worker = pick_shift(s1)
su2_worker = pick_shift(s2)

# once schedule is generated, go through schedule and check if individual is working twice in the same day
# if they are not working the day already or are working the shift before or after.
schedule = {}

schedule[0] = [m1_worker, '6am-10am']
schedule[1] = [m2_worker, '10am-2pm']
schedule[2] = [m3_worker, '2pm-6pm']
schedule[3] = [m4_worker, '6pm-10pm']

schedule[4] = [t1_worker, '6am-10am']
schedule[5] = [t2_worker, '10am-2pm']
schedule[6] = [t3_worker, '2pm-6pm']
schedule[7] = [t4_worker, '6pm-10pm']

schedule[8] = [w1_worker, '6am-10am']
schedule[9] = [w2_worker, '10am-2pm']
schedule[10] = [w3_worker, '2pm-6pm']
schedule[11] = [w4_worker, '6pm-10pm']

schedule[12] = [th1_worker, '6am-10am']
schedule[13] = [th2_worker, '10am-2pm']
schedule[14] = [th3_worker, '2pm-6pm']
schedule[15] = [th4_worker, '6pm-10pm']

schedule[16] = [f1_worker, '6am-10am']
schedule[17] = [f2_worker, '10am-2pm']
schedule[18] = [f3_worker, '2pm-6pm']
schedule[19] = [f4_worker, '6pm-10pm']

schedule[20] = [s1_worker, '7am-12pm']
schedule[21] = [s2_worker, '12pm-5pm']

schedule[22] = [su1_worker, '7am-12pm']
schedule[23] = [su2_worker, '12pm-5pm']

# create dictionary of names and worker ID
names_results = read_query(connection, get_names)
id_to_name = {}
create_student_dict(id_to_name, names_results)

# change workers to names by for loop going through list
for i in range(total_shifts):
    for id in id_to_name:
        if schedule[i][0] == id:
            schedule[i][0] = id_to_name[id]
```



```


# create file that shows the schedule
# in print schedule file

# get date for the schedule
date = read_query(connection, get_week)

```


database.py


```
''' Database set up for project
include tables: STUDENT, STUDENT_AVAILABILITY'''

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)
"""

insert_s1 = 'INSERT INTO student(id, first_name, last_name, department, class) VALUES (001, "Vanshita", "Malhotra", "Front Desk", "Sophomore");'
insert_s2 = 'INSERT INTO student(id, first_name, last_name, department, class) VALUES (002, "Michael", "Cross", "Front Desk", "Sophomore");'
insert_s3 = 'INSERT INTO student(id, first_name, last_name, department, class) VALUES (003, "Sara", "Zorzan", "Front Desk", "Sophomore");'
insert_s4 = 'INSERT INTO student(id, first_name, last_name, department, class) VALUES (004, "Giada", "Zorzan", "Front Desk", "Junior");'
insert_s5 = 'INSERT INTO student(id, first_name, last_name, department, class) VALUES (005, "Nico", "Teynie", "Front Desk", "Senior");'
insert_s6 = 'INSERT INTO student(id, first_name, last_name, department, class) VALUES (006, "Aaron", "Mantilla", "Front Desk", "Senior");'
insert_s7 = 'INSERT INTO student(id, first_name, last_name, department, class) VALUES (007, "Priyam", "Shah", "Front Desk", "Graduate");'
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');"""
insert_s2_avail = """INSERT INTO student_availability(id, availability, week_beginning) VALUES(002,
    '{
    "Monday" : {
        "start" : "0800",
        "end"   : "1300"},
    "Tuesday" : {
        "start" : "1500",
        "end"   : "2200"},
    "Wednesday": {
        "start" : "1500",
        "end"   : "1900"},
    "Thursday" : {
        "start" : "null",
        "end"   : "null"},
    "Friday" : {
        "start" : "0800",
        "end"   : "1800"},
    "Saturday" : {
        "start" : "0700",
        "end"   : "1500"},
    "Sunday" : {
        "start" : "1200",
        "end"   : "2000"}
    }',
    '2022-12-05');"""
insert_s3_avail = """INSERT INTO student_availability(id, availability, week_beginning) VALUES(003,
    '{"Monday"   : {
        "start" : "null",
        "end"   : "null"},
    "Tuesday"  : {
        "start" : "0900",
        "end"   : "2200"},
    "Wednesday": {
        "start" : "1000",
        "end"   : "2000"},
    "Thursday" : {
        "start" : "0800",
        "end"   : "1400"},
    "Friday"   : {
        "start" : "1500",
        "end"   : "2300"},
    "Saturday" : {
        "start" : "0600",
        "end"   : "1300"},
    "Sunday"   : {
        "start" : "0600",
```

```
        "end"   : "1200"}
    }', '2022-12-05');"""
insert_s4_avail = """INSERT INTO student_availability(id, availability,
week_beginning) VALUES(004,
    '{"Monday"   : {
        "start" : "1300",
        "end"   : "2200"},
    "Tuesday"  : {
        "start" : "0600",
        "end"   : "1200"},
    "Wednesday": {
        "start" : "1000",
        "end"   : "1700"},
    "Thursday" : {
        "start" : "0500",
        "end"   : "1300"},
    "Friday"   : {
        "start" : "null",
        "end"   : "null"},
    "Saturday" : {
        "start" : "0700",
        "end"   : "1300"},
    "Sunday"   : {
        "start" : "null",
        "end"   : "null"}
    }', '2022-12-05');"""
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');"""
insert_s6_avail = """INSERT INTO student_availability(id, availability,
week_beginning) VALUES(006,
    '{"Monday"   : {
        "start" : "1700",
        "end"   : "2200"},
    "Tuesday"  : {
        "start" : "0600",
        "end"   : "1800"},
    "Wednesday": {
        "start" : "0600",
```



```
        "end"   : "1400"},
    "Thursday" : {
        "start" : "1300",
        "end"   : "2200"},
    "Friday"   : {
        "start" : "0600",
        "end"   : "1800"},
    "Saturday" : {
        "start" : "0700",
        "end"   : "1700"},
    "Sunday"   : {
        "start" : "0700",
        "end"   : "1700"}
    }', '2022-12-05');"""
insert_s7_avail = """INSERT INTO student_availability(id, availability,
week_beginning) VALUES(007,
    '{"Monday"   : {
        "start" : "0600",
        "end"   : "1500"},
    "Tuesday"  : {
        "start" : "1300",
        "end"   : "2300"},
    "Wednesday": {
        "start" : "1300",
        "end"   : "2300"},
    "Thursday" : {
        "start" : "1300",
        "end"   : "2300"},
    "Friday"   : {
        "start" : "1700",
        "end"   : "2300"},
    "Saturday" : {
        "start" : "null",
        "end"   : "null"},
    "Sunday"   : {
        "start" : "0700",
        "end"   : "1200"}
    }', '2022-12-05');"""
insert_s8_avail = """INSERT INTO student_availability(id, availability,
week_beginning) VALUES(008,
    '{"Monday"   : {
        "start" : "1800",
        "end"   : "2300"},
    "Tuesday"  : {
        "start" : "1000",
        "end"   : "1400"},
    "Wednesday": {
        "start" : "0600",
        "end"   : "1400"},
    "Thursday" : {
        "start" : "0600",
        "end"   : "1800"},
    "Friday"   : {
        "start" : "1300",
        "end"   : "2200"},
    "Saturday" : {
        "start" : "0700",
        "end"   : "1200"},
    "Sunday"   : {
        "start" : "1200",
```



```
        "end"   : "1700"}
    }', '2022-12-05');"""
```

queries.py


```
''' Queries for the project

SHIFTS
MTWThF: 6-10, 10-2, 2-6, 6-10
SSu: 7-12, 12-5
'''

# these shifts are hard coded
# returns id that is available for the shift
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;
"""
t_first_shift_avail = """
SELECT ID
FROM student_availability
WHERE CAST(JSON_EXTRACT(availability, "$.Tuesday.start") AS DECIMAL) <= 600
AND CAST(JSON_EXTRACT(availability, "$.Tuesday.end") AS DECIMAL) >= 1000;
"""
w_first_shift_avail = """
SELECT ID
FROM student_availability
WHERE CAST(JSON_EXTRACT(availability, "$.Wednesday.start") AS DECIMAL) <= 600
AND CAST(JSON_EXTRACT(availability, "$.Wednesday.end") AS DECIMAL) >= 1000;
"""
th_first_shift_avail = """
SELECT ID
FROM student_availability
WHERE CAST(JSON_EXTRACT(availability, "$.Thursday.start") AS DECIMAL) <= 600
AND CAST(JSON_EXTRACT(availability, "$.Thursday.end") AS DECIMAL) >= 1000;
"""
f_first_shift_avail = """
SELECT ID
FROM student_availability
WHERE CAST(JSON_EXTRACT(availability, "$.Friday.start") AS DECIMAL) <= 600
AND CAST(JSON_EXTRACT(availability, "$.Friday.end") AS DECIMAL) >= 1000;
"""
s_first_shift_avail = """
SELECT ID
FROM student_availability
WHERE CAST(JSON_EXTRACT(availability, "$.Saturday.start") AS DECIMAL) <= 700
AND CAST(JSON_EXTRACT(availability, "$.Saturday.end") AS DECIMAL) >= 1200;
"""
su_first_shift_avail = """
SELECT ID
FROM student_availability
WHERE CAST(JSON_EXTRACT(availability, "$.Sunday.start") AS DECIMAL) <= 700
AND CAST(JSON_EXTRACT(availability, "$.Sunday.end") AS DECIMAL) >= 1200;
"""

m_second_shift_avail = """
SELECT ID
FROM student_availability
WHERE CAST(JSON_EXTRACT(availability, "$.Monday.start") AS DECIMAL) <= 1000
AND CAST(JSON_EXTRACT(availability, "$.Monday.end") AS DECIMAL) >= 1400;
"""
t_second_shift_avail = """
```



```
SELECT ID
FROM student_availability
WHERE CAST(JSON_EXTRACT(availability, "$.Tuesday.start") AS DECIMAL) <= 1000
AND CAST(JSON_EXTRACT(availability, "$.Tuesday.end") AS DECIMAL) >= 1400;
"""
w_second_shift_avail = """
SELECT ID
FROM student_availability
WHERE CAST(JSON_EXTRACT(availability, "$.Wednesday.start") AS DECIMAL) <=
1000
AND CAST(JSON_EXTRACT(availability, "$.Wednesday.end") AS DECIMAL) >= 1400;
"""
th_second_shift_avail = """
SELECT ID
FROM student_availability
WHERE CAST(JSON_EXTRACT(availability, "$.Thursday.start") AS DECIMAL) <= 1000
AND CAST(JSON_EXTRACT(availability, "$.Thursday.end") AS DECIMAL) >= 1400;
"""
f_second_shift_avail = """
SELECT ID
FROM student_availability
WHERE CAST(JSON_EXTRACT(availability, "$.Friday.start") AS DECIMAL) <= 1000
AND CAST(JSON_EXTRACT(availability, "$.Friday.end") AS DECIMAL) >= 1400;
"""
s_second_shift_avail = """
SELECT ID
FROM student_availability
WHERE CAST(JSON_EXTRACT(availability, "$.Saturday.start") AS DECIMAL) <= 1200
AND CAST(JSON_EXTRACT(availability, "$.Saturday.end") AS DECIMAL) >= 1700;
"""
su_second_shift_avail = """
SELECT ID
FROM student_availability
WHERE CAST(JSON_EXTRACT(availability, "$.Sunday.start") AS DECIMAL) <= 1200
AND CAST(JSON_EXTRACT(availability, "$.Sunday.end") AS DECIMAL) >= 1700;
"""

m_third_shift_avail = """
SELECT ID
FROM student_availability
WHERE CAST(JSON_EXTRACT(availability, "$.Monday.start") AS DECIMAL) <= 1400
AND CAST(JSON_EXTRACT(availability, "$.Monday.end") AS DECIMAL) >= 1800;
"""
t_third_shift_avail = """
SELECT ID
FROM student_availability
WHERE CAST(JSON_EXTRACT(availability, "$.Tuesday.start") AS DECIMAL) <= 1400
AND CAST(JSON_EXTRACT(availability, "$.Tuesday.end") AS DECIMAL) >= 1800;
"""
w_third_shift_avail = """
SELECT ID
FROM student_availability
WHERE CAST(JSON_EXTRACT(availability, "$.Wednesday.start") AS DECIMAL) <=
1400
AND CAST(JSON_EXTRACT(availability, "$.Wednesday.end") AS DECIMAL) >= 1800;
"""
th_third_shift_avail = """
SELECT ID
FROM student_availability
WHERE CAST(JSON_EXTRACT(availability, "$.Thursday.start") AS DECIMAL) <= 1400
```



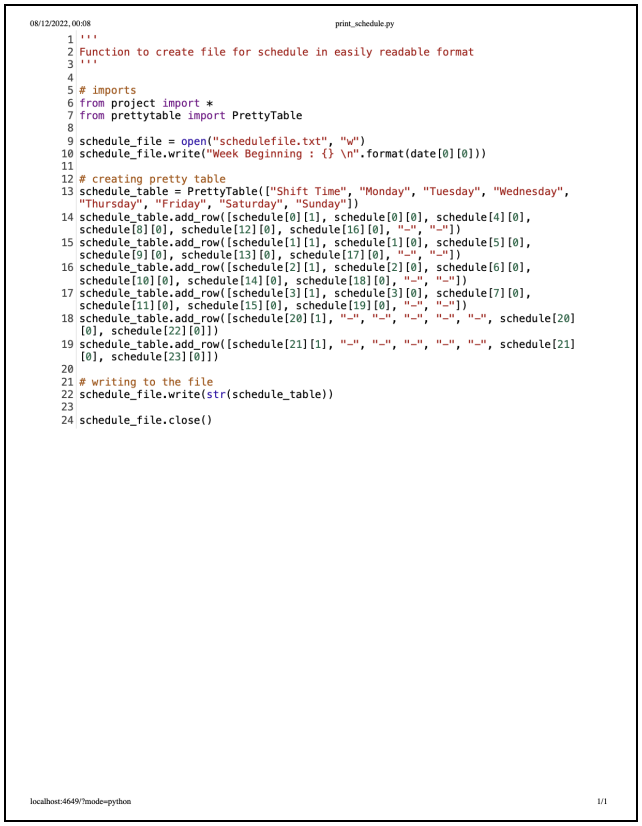
```
AND CAST(JSON_EXTRACT(availability, "$.Thursday.end") AS DECIMAL) >= 1800;
"""
f_third_shift_avail = """
SELECT ID
FROM student_availability
WHERE CAST(JSON_EXTRACT(availability, "$.Friday.start") AS DECIMAL) <= 1400
AND CAST(JSON_EXTRACT(availability, "$.Friday.end") AS DECIMAL) >= 1800;
"""

m_fourth_shift_avail = """
SELECT ID
FROM student_availability
WHERE CAST(JSON_EXTRACT(availability, "$.Monday.start") AS DECIMAL) <= 1800
AND CAST(JSON_EXTRACT(availability, "$.Monday.end") AS DECIMAL) >= 2200;
"""
t_fourth_shift_avail = """
SELECT ID
FROM student_availability
WHERE CAST(JSON_EXTRACT(availability, "$.Tuesday.start") AS DECIMAL) <= 1800
AND CAST(JSON_EXTRACT(availability, "$.Tuesday.end") AS DECIMAL) >= 2200;
"""
w_fourth_shift_avail = """
SELECT ID
FROM student_availability
WHERE CAST(JSON_EXTRACT(availability, "$.Wednesday.start") AS DECIMAL) <=
1800
AND CAST(JSON_EXTRACT(availability, "$.Wednesday.end") AS DECIMAL) >= 2200;
"""
th_fourth_shift_avail = """
SELECT ID
FROM student_availability
WHERE CAST(JSON_EXTRACT(availability, "$.Thursday.start") AS DECIMAL) <= 1800
AND CAST(JSON_EXTRACT(availability, "$.Thursday.end") AS DECIMAL) >= 2200;
"""
f_fourth_shift_avail = """
SELECT ID
FROM student_availability
WHERE CAST(JSON_EXTRACT(availability, "$.Friday.start") AS DECIMAL) <= 1800
AND CAST(JSON_EXTRACT(availability, "$.Friday.end") AS DECIMAL) >= 2200;
"""

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

get_week = """
SELECT DATE_FORMAT(week_beginning, '%Y-%m-%d') as date
FROM student_availability
GROUP BY 1;
"""
```

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",
   "Thursday", "Friday", "Saturday", "Sunday"])
14 schedule_table.add_row([schedule[0][1], schedule[0][0], schedule[4][0],
   schedule[8][0], schedule[12][0], schedule[16][0], "-", "-"])
15 schedule_table.add_row([schedule[1][1], schedule[1][0], schedule[5][0],
   schedule[9][0], schedule[13][0], schedule[17][0], "-", "-"])
16 schedule_table.add_row([schedule[2][1], schedule[2][0], schedule[6][0],
   schedule[10][0], schedule[14][0], schedule[18][0], "-", "-"])
17 schedule_table.add_row([schedule[3][1], schedule[3][0], schedule[7][0],
   schedule[11][0], schedule[15][0], schedule[19][0], "-", "-"])
18 schedule_table.add_row([schedule[20][1], "-", "-", "-", "-", "-", schedule[20]
   [0], schedule[22][0]])
19 schedule_table.add_row([schedule[21][1], "-", "-", "-", "-", "-", schedule[21]
   [0], schedule[23][0]])
20
21 # writing to the file
22 schedule_file.write(str(schedule_table))
23
24 schedule_file.close()
```

localhost:4649/?mode=python 1/1