USE CASE STUDY REPORT

Group No: 1
Aditi Thakur and Ayush Venkatesh

Executive Summary:

Due to COVID-19, Northeastern University is required to regularly test and vaccinate its students. Since there are a very large number of students at Northeastern University, it is necessary to create a database to store the details of each student and track each student's status on testing and vaccination. In addition to being able to store a large number of records, our database can easily update and retrieve relevant details of a student. One can also perform analytics using our database to collect various statistics. In addition, one can enhance our database by updating and adding more tables and gaining additional insights using analytics.

The requirements of our database were collected based on our experience getting tested at Cabot testing Center. The requirements included personal details about each student as well as details about testing and vaccination for each student. We started with modelling the EER and UML diagram, followed by mapping of a conceptual model to a relational model with the primary keys and foreign keys. We then implemented a relational database on MySQL as well as a non relational database on MongoDB. In addition to the database, we also used Python to connect to the database and computed various statistics on students, demonstrating analytics capabilities as well.

Finally, while our database is a success, there is room for improvement, which is discussed in the conclusion of this report.

I. Introduction

Northeastern University is a large university in Boston with over 27,000 students. In addition, Boston is a hub for many companies across various industries from Technology to Healthcare. Northeastern University has a large co-op program with employers from many companies visiting to conduct interviews. In addition, international students from many different countries come to Northeastern to study. Thus, in such a scenario, there are many different ways in which COVID could spread among students. For example, an employer might catch it in their workplace and then while conducting an in-person interview with a student, pass it on; An international student might potentially catch it in their home country and then spread it to another student on campus.

In such a scenario, it is important to test everyone on campus, and isolate people who test positive. Given the large number of people (students and otherwise) on campus, using pen and paper to track each student becomes very cumbersome.

Therefore, a relational database is ideal for such a situation. A relational database has no trouble storing the details of all students as well as tracking details on testing and vaccination. In addition, such a centralized system allows for easy access of data anywhere on campus or remotely. Multiple users can work on the database at anytime. Access (full or partial) can be given or denied to some users. For example, someone who is not in charge of collecting data can be denied write and modify access and only given read access. In addition, using Python, one can compute various descriptive statistics as well.

In our database, we divide students into three categories: Frontline, Non-Frontline and Regular. Frontline and Non-Frontline students are those who work on campus at the testing center in frontline and non frontline roles respectively. Regular students are those who do not work, but only study. Therefore, there are three tables for each category, called regular_student, frontline_student and non_frontline_student. In each table, we store the following columns: Name, Email, NUID, Date of Birth (DOB) and pre-existing condition. The last column indicates if a student has a pre-existing condition. Since we need to test each student, there are tables in the database called testing_r (regular), testing_f (frontline) and testing_n (non frontline) corresponding to the tables regular_student, frontline_student and non_frontline_student. When a student gets tested, a barcode is generated for their nasal swab. So, for the testing_r, it contains the details regular_nuid and barcode. Similar columns are in the other testing tables. Each test has a result: positive or negative. Hence, there are three results tables: result_r, result_f and result_n. Each result table stores the NUID and the result, which is a Yes/No. Vaccination is done in two phases Phase 1 and Phase 2. For each phase there are 3 tables, regular, frontline and non-frontline. For example: phase1_r, phase1_f, and phase1_n. Each table stores the NUID and a column called eligiblity_check. Only students (of any category) who have pre-existing conditions are eligible and must compulsorily get vaccinated in Phase 1. Otherwise, they can only get vaccinated in phase 2. Hence, for phase 1 tables, eligiblity_check is ‘Yes’ if the student has pre-existing conditions, ‘No’ otherwise. For phase 2, only students who do not have pre-exisitng conditions are allowed. Hence eligibility_check is populated in the opposite way as phase 1.

We also implement a part of the database in MongoDB and perform queries there as well.

II. Conceptual Data Modelling

1. EER Diagram

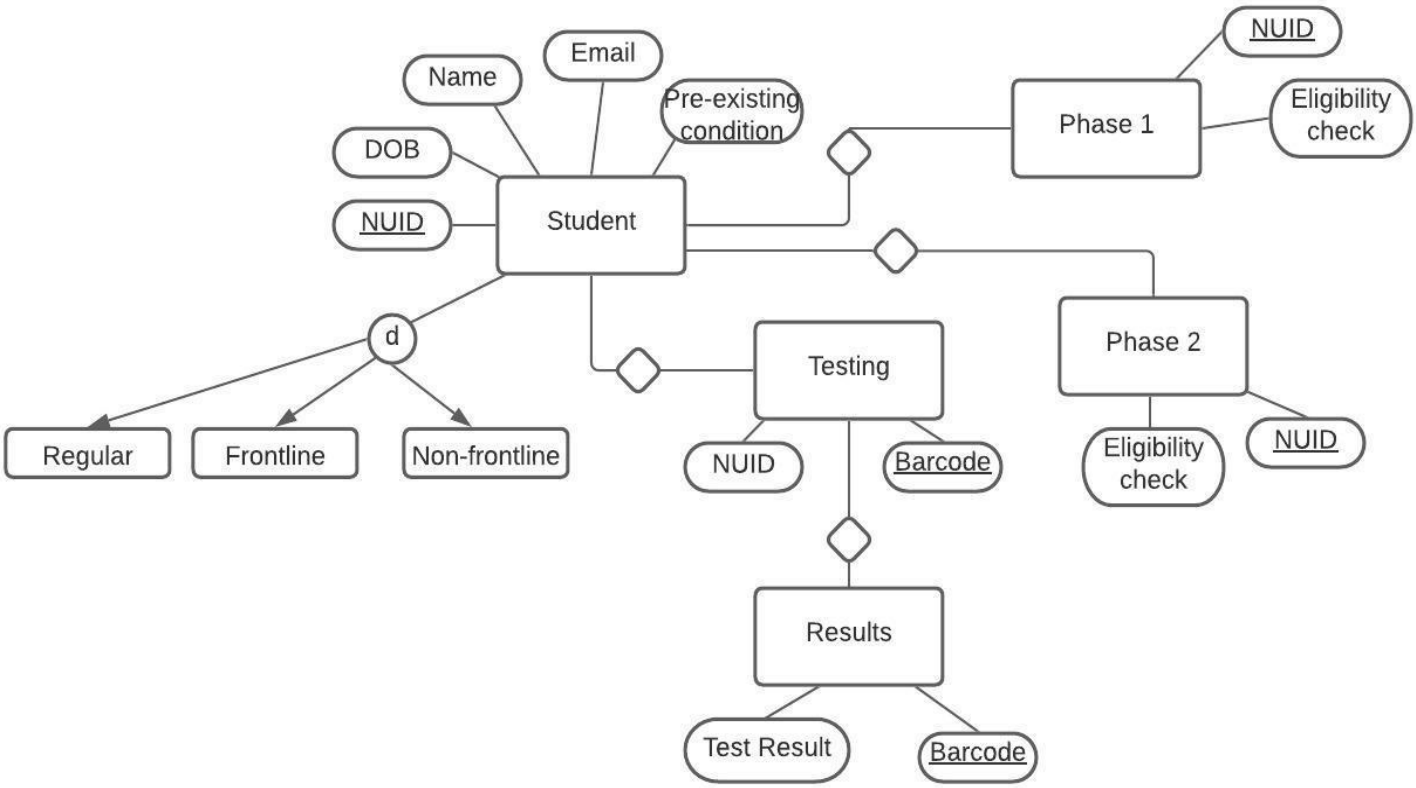

Fig 1: EER diagram for Cabot Testing Center

2. UML Diagram

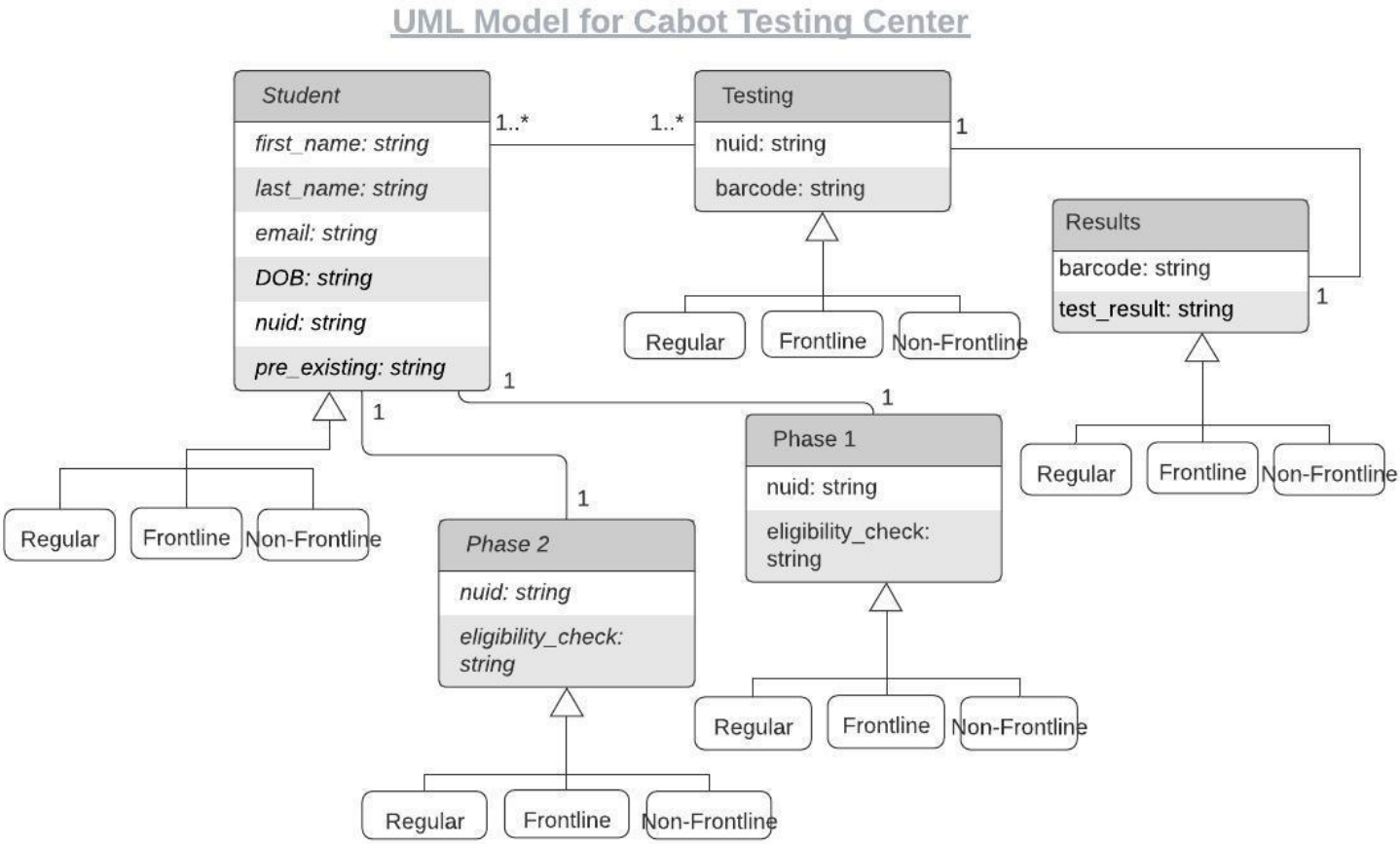

Fig 2: UML diagram for Cabot Testing Center

III. Mapping of a conceptual model to a relational model

Primary Keys in Bold, Foreign in Italic

Regular_Student (name, **NUID**, email, DOB, pre-existing_condition)

Frontline_Student (name, **NUID**, email, DOB, pre-existing_condition)

Non_Frontline_Student (name, **NUID**, email, DOB, pre-existing_condition)

Testing_r (*NUID*, **barcode**)

Testing_ f(*NUID*, **barcode**)

Testing_n(*NUID*, **barcode**)

Results_r (**barcode**, test_result)

Results_f (**barcode**, test_result)

Results_n (**barcode**, test_result)

Phase1_r (***NUID***, eligibility_check)

Phase1_f (***NUID***, eligibility_check)

Phase1_n (***NUID***, eligibility_check)

Phase2_r (***NUID***, eligibility_check)

Phase2_f (***NUID***, eligibility_check)

Phase2_n (***NUID***, eligibility_check)

IV. Implementation of relational model using MySQL and MongoDB

MySQL Implementation:

1. Find the name, email and NUID for students who have pre-existing conditions.

```
#regular
SELECT first_name, last_name, email, regular_nuid
FROM regular_student
WHERE pre_existing = "Yes";
#frontline
SELECT first_name, last_name, email, frontline_nuid
FROM frontline_student
WHERE pre_existing = "Yes";
#non frontline
SELECT first_name, last_name, email, non_frontline_nuid
FROM non_frontline_student
WHERE pre_existing = "Yes";
```

2. Find the name, email and NUID of all students who test positive.

```
SELECT f.frontline_nuid, f.first_name, f.last_name, f.email
FROM frontline_student f, testing_f t, results_f r
WHERE f.frontline_nuid=t.frontline_nuid and t.barcode = r.barcode and r.result='YES';

SELECT rs.regular_nuid, rs.first_name, rs.last_name, rs.email
FROM regular_student rs, testing_r t, results_r r
WHERE rs.regular_nuid=t.regular_nuid and t.barcode = r.barcode and r.result='YES';

SELECT n.non_frontline_nuid, n.first_name, n.last_name, n.email
FROM non_frontline_student n, testing_n t, results_n r
WHERE n.non_frontline_nuid=t.non_frontline_nuid and t.barcode = r.barcode and
r.result='YES';
```

3. Get the total number of students vaccinated from all the phase tables, and check if it is equal to the total number of students in the student tables. That way everyone is getting vaccinated.

```
#From all phase tables
SELECT (
(select count(regular_nuid)
from phase1_r
where eligibility_check="Y")
+
(select count(frontline_nuid)
from phase1_f
where eligibility_check="Y")
+
(select count(non_frontline_nuid)
from phase1_n
where eligibility_check="Y")
+
(select count(regular_nuid)
from phase2_r
where eligibility_check="Y")
+
(select count(frontline_nuid)
from phase2_f
where eligibility_check="Y")
+
(select count(non_frontline_nuid)
from phase2_n
where eligibility_check="Y")
)
as total

#From student tables
SELECT (
(select count(regular_nuid) from regular_student)
+
(select count(FRONTLINE_nuid) from FRONTLINE_student)
+
(select count(non_frontline_nuid) from non_frontline_student)
)
AS total
```

4. Check if there are any students in Phase 1 who are not supposed to be vaccinated in phase 1 i.e. with eligibility_check = 'N'. Delete all such records.

```
#Some records in phase in phase1_frontline have eligiblity_check = 'N', hence delete them
SELECT * FROM phase1_f where eligibility_check = 'N'
DELETE FROM phase1_F where eligibility_check = 'N'

#None in Phase1_ regular
SELECT * FROM phase1_r where eligibility_check = 'N'

#None in Phase1_non_Frontline
SELECT * FROM phase1_n where eligibility_check = 'N'
```

5. Get all students who have positive test cases. Group by pre-existing condition. That way, we know the number of students who test positive who are eligible for phase 1, and the number of students who don't test positive who are eligible for phase 1.

```
SELECT f.pre_existing as 'Pre Existing Condition', count(*) as 'Number of positive'
FROM frontline_student f, testing_f t, results_f r
WHERE f.frontline_nuid=t.frontline_nuid and t.barcode = r.barcode and r.test_result="Yes"
GROUP BY f.pre_existing;

SELECT rs.pre_existing as 'Pre Existing Condition', count(*) as 'Number of positive'
FROM regular_student rs, testing_r t, results_r r
WHERE rs.regular_nuid=t.regular_nuid and t.barcode = r.barcode and r.test_result="Yes"
GROUP BY rs.pre_existing;

SELECT n.pre_existing as 'Pre Existing Condition', count(*) as 'Number of positive'
FROM non_frontline_student n, testing_n t, results_n r
WHERE n.non_frontline_nuid=t.non_frontline_nuid and t.barcode = r.barcode and
r.test_result="Yes"
GROUP BY n.pre_existing;
```

MongoDB Implementation

1. Name, email and NUID of students who have pre-existing conditions.

```
db.regular_student.aggregate([
```

```
  { $match: {pre_existing: "Yes"}}]);
db.frontline_student.aggregate([
  { $match: {pre_existing: "Yes"}}]);
db.non_frontline_student.aggregate([
  { $match: {pre_existing: "Yes"}}]);
```

2. Check if there are any students in Phase 1 who are not supposed to be vaccinated in phase 1 i.e. with eligibility_check = 'N'. Delete all such records.

```
db.phase1_r.aggregate([
  { $group: { _id: "$r_nuid", total: { $sum:1}}}
  ]);
db.phase2_r.aggregate([
   { $group: { _id: "$r_nuid", total: { $sum:1}}}
  ]);
db.phase1_r.aggregate([
  { $match: {eligibility: "No"}}])
```

3. List of all frontline students over the age of 18.

```
var condition = {
  $project:{
    first_name:1,
    last_name:1,
    age:{
      $floor: {
        $divide: [
          {$subtract: [ new Date(), "$DOB" ]},
          (365 * 24 * 60 * 60 * 1000)
        ]
      }
    }
  }
},
match = {
  $match: {age: {$gte:18}}
};
db.frontline_student.aggregate([condition, match])
```

V. Analytics

We used Python to access the database using the mysql.connector library. We executed the query using the cursor function. We did some additional processing of the data if necessary. For example, to calculate age, we retrieved the date of birth, and used Python to compute the present age of students. We computed various statistics and created charts using the Matplotlib library.

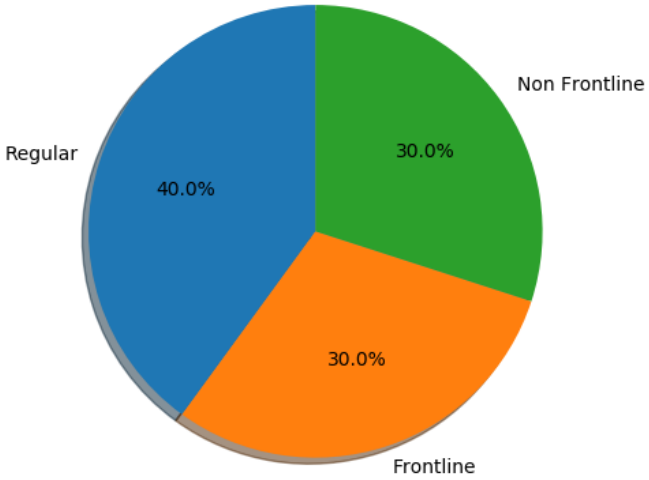

Fig 3: Distribution of students among Frontline, Non-Frontline and Regular

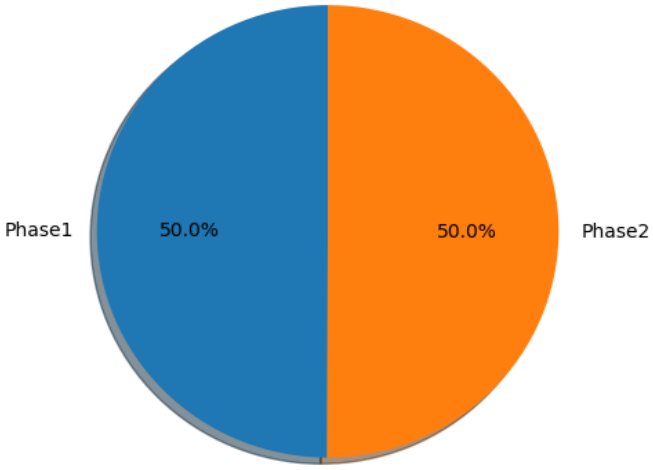

Fig 4: Distribution of students getting vaccinated in Phase 1 and Phase 2

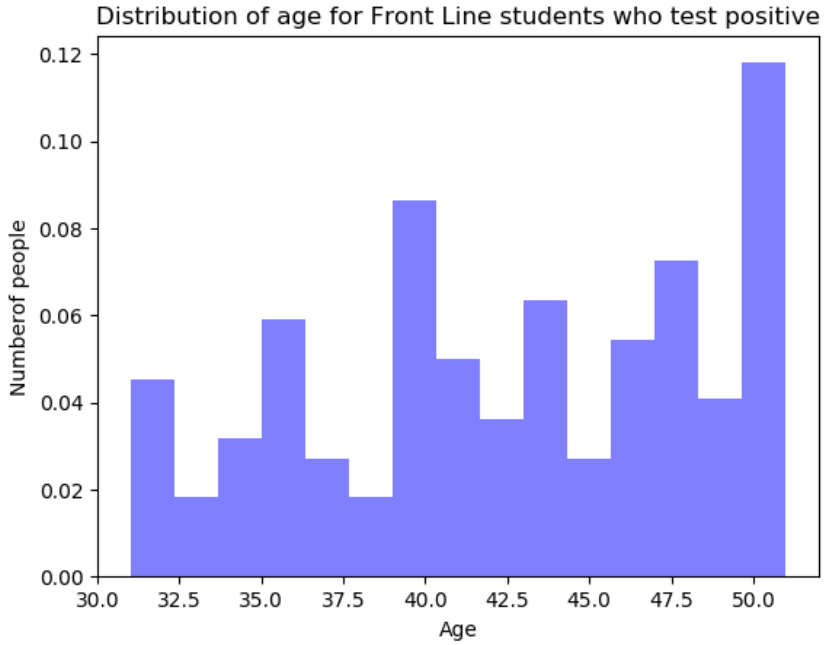

Fig 5: Distribution of age for frontline students who test positive

VI. Conclusion and Recommendations

Our database is fully functional. One can work with the database as well as perform analytics using Python. However, there are improvements that can be made to our database. The following are some suggestions:

1.) The phase tables are populated using the INSERT INTO statements. Since these are done manually, a better way for future implementation is using triggers.

2.) We have separate tables for each category: Frontline, Regular, and Non-Frontline. A better method to reduce tables by combining data would be more ideal.

3.) Right now, our database can only track 1 round of testing for students and assumes that everyone gets tested. For future implementations, we can add functionality to check if students are getting tested on a regular basis, i.e., if they are compliant.

4.) Our database does not track remote students, who do not need to be tested, but need to be vaccinated. Hence for future implementation, we can add functionality to check if students got vaccinated outside the campus (including out of the country) and update our database.

5.) Students who test positive will need to be quarantined, hence we can add functionality to track students who are being quarantined and update their status once they are cured of COVID-19.

VII. Appendix (all SQL/MongoDB code to create tables and Python code)

MongoDB:

```
db.getCollectionNames();
db.createCollection("regular_student");
db.createCollection("frontline_student");
db.createCollection("non_frontline_student");
db.createCollection("phase1_r");
db.createCollection("phase1_f");
db.createCollection("phase1_n");
db.createCollection("phase2_r");
db.createCollection("phase2_f");
db.createCollection("phase2_n");
db.regular_student.insertMany(
	[
		{
			r_nuid: "002567890",
			first_name: "Radhika",
			last_name: "Sharma",
			DOB: "1995-09-26",
			email: "radhika_sharma.123@northeastern.edu",
			pre_existing: "Yes"
		},
		{
			r_nuid: "765253085",
			first_name: "Rachel",
			last_name: "Christopher",
			DOB: "1990-02-16",
			email: "Rachel_Christopher.123@gmail.com",
			pre_existing: "No"
		},
		{
			r_nuid: "002508264",
			first_name: "Katlyn",
			last_name: "Bert",
			DOB: "1996-05-21",
			email: "katlyn_bert@northeastern.edu",
```

```
                pre_existing: "Yes"
            },
            {
                r_nuid: "729502629",
                first_name: "Adam",
                last_name: "Jonas",
                DOB: "1993-07-1",
                email: "adam5jonas@gmail.com",
                pre_existing: "No"
            },
            {
                r_nuid: "926046173",
                first_name: "Fathima",
                last_name: "Sheik",
                DOB: "1990-02-16",
                email: "Fathima_Sheik.123@gmail.com",
                pre_existing: "Yes"
            }
        ]
);
db.frontline_student.insertMany(
        [
            {
                f_nuid: "082567280",
                first_name: "Shami",
                last_name: "Anand",
                DOB: new Date("1993-10-16"),
                email: "shamianand@northeastern.edu",
                pre_existing: "Yes"
            },
            {
                f_nuid: "765827685",
                first_name: "Ray",
                last_name: "Croc",
                DOB: new Date("1996-02-16"),
                email: "raycroc3@gmail.com",
                pre_existing: "No"
            },
```

```
		{
			f_nuid: "102925264",
			first_name: "Natasha",
			last_name: "Hammock",
			DOB: new Date("1993-05-11"),
			email: "natashahammock@northeastern.edu",
			pre_existing: "Yes"
		},
		{
			f_nuid: "729509279",
			first_name: "Martha",
			last_name: "Bob",
			DOB: new Date("1993-07-17"),
			email: "marthabob@gmail.com",
			pre_existing: "No"
		},
		{
			f_nuid: "926071933",
			first_name: "Fred",
			last_name: "Jones",
			DOB: new Date("1996-02-16"),
			email: "fredjones@gmail.com",
			pre_existing: "Yes"
		}
	]
);
db.non_frontline_student.insertMany(
	[
		{
			n_nuid: "082916380",
			first_name: "Sam",
			last_name: "Richard",
			DOB: "1993-7-16",
			email: "samrichard@northeastern.edu",
			pre_existing: "Yes"
		},
		{
			f_nuid: "765891675",
```

```
                first_name: "Rita",
                last_name: "Felix",
                DOB: "1997-02-6",
                email: "ritafelix3@gmail.com",
                pre_existing: "No"
            },
            {
                f_nuid: "108915264",
                first_name: "Alex",
                last_name: "Richmond",
                DOB: "1993-03-9",
                email: "alexrichmond@northeastern.edu",
                pre_existing: "Yes"
            },
            {
                f_nuid: "729590159",
                first_name: "Cody",
                last_name: "Zen",
                DOB: "1993-12-7",
                email: "codyzen.com",
                pre_existing: "No"
            },
            {
                f_nuid: "928916933",
                first_name: "Anukriti",
                last_name: "Patel",
                DOB: "1996-03-19",
                email: "anukritipatel@gmail.com",
                pre_existing: "Yes"
            }
        ]
);
db.phase1_r.insertMany(
  [
    {
      r_nuid: "002567890",
      eligibility: "Yes"
    },
```

```
    {
      r_nuid: "765253085",
      eligibility: "No"
    },
    {
      r_nuid: "002508264",
      eligibility: "No"
    }
  ]
);
db.phase1_f.insertMany(
  [
    {
      f_nuid: "082567280",
      eligibility: "Yes"
    },
    {
      f_nuid: "765827685",
      eligibility: "No"
    }
  ]
);
db.phase1_n.insertMany(
  [
    {
      n_nuid: "082916380",
      eligibility: "Yes"
    },
    {
      n_nuid: "765891675",
      eligibility: "No"
    },
    {
      n_nuid: "108915264",
      eligibility: "Yes"
    }
  ]
);
```

```
db.phase2_r.insertMany(
  [
    {
      r_nuid: "729502629",
      eligibility: "Yes"
    },
    {
      r_nuid: "926046173",
      eligibility: "Yes"
    }
  ]
);
db.phase2_f.insertMany(
  [
    {
      f_nuid: "102925264",
      eligibility: "No"
    },
    {
      f_nuid: "729509279",
      eligibility: "Yes"
    },
    {
      f_nuid: "926071933",
      eligibility: "Yes"
    }
  ]
);
db.phase2_n.insertMany(
  [
    {
      n_nuid: "729590159",
      eligibility: "Yes"
    },
    {
      n_nuid: "928916933",
      eligibility: "No"
    }]);
```

SQL:

```
CREATE TABLE regular_student (
regular_nuid VARCHAR(200) PRIMARY KEY,
first_name VARCHAR(200),
last_name VARCHAR(200),
email VARCHAR(200),
DOB VARCHAR(200),
pre_existing VARCHAR(10)
);

SELECT * FROM regular_student
CREATE TABLE frontline_student (
frontline_nuid VARCHAR(200) PRIMARY KEY,
first_name VARCHAR(200),
last_name VARCHAR(200),
email VARCHAR(200),
DOB VARCHAR(200),
pre_existing VARCHAR(10)
);
SELECT * FROM frontline_student
CREATE TABLE non_frontline_student (
non_frontline_nuid VARCHAR(200) PRIMARY KEY,
first_name VARCHAR(200),
last_name VARCHAR(200),
email VARCHAR(200),
DOB VARCHAR(200),
pre_existing VARCHAR(10)
);
SELECT * FROM non_frontline_student


CREATE TABLE testing_r (
barcode VARCHAR(200) PRIMARY KEY,
regular_nuid VARCHAR(200),
FOREIGN KEY(regular_nuid) references regular_student(regular_nuid)
);
CREATE TABLE testing_f (
```

```
barcode VARCHAR(200) PRIMARY KEY,
frontline_nuid VARCHAR(200),
FOREIGN KEY(frontline_nuid) references frontline_student(frontline_nuid)
);
CREATE TABLE testing_n (
barcode VARCHAR(200) PRIMARY KEY,
non_frontline_nuid VARCHAR(200),
FOREIGN KEY(non_frontline_nuid) references non_frontline_student(non_frontline_nuid)
);
CREATE TABLE results_r (
barcode VARCHAR(200) PRIMARY KEY,
test_result VARCHAR(100)
);
CREATE TABLE results_f (
barcode VARCHAR(200) PRIMARY KEY,
test_result VARCHAR(100)
);
CREATE TABLE results_n (
barcode VARCHAR(200) PRIMARY KEY,
test_result VARCHAR(100)
);
CREATE TABLE phase1_r (
regular_nuid VARCHAR(200) PRIMARY KEY,
FOREIGN KEY(regular_nuid) references regular_student(regular_nuid),
eligibility_check VARCHAR(100)
);
CREATE TABLE phase1_f (
frontline_nuid VARCHAR(200) PRIMARY KEY,
FOREIGN KEY(frontline_nuid) references frontline_student(frontline_nuid),
eligibility_check VARCHAR(100)
);
CREATE TABLE phase1_n (
non_frontline_nuid VARCHAR(200) PRIMARY KEY,
FOREIGN KEY(non_frontline_nuid) references non_frontline_student(non_frontline_nuid),
eligibility_check VARCHAR(100)
);
CREATE TABLE phase2_r (
regular_nuid VARCHAR(200) PRIMARY KEY,
```

```
FOREIGN KEY(regular_nuid) references regular_student(regular_nuid),
eligibility_check VARCHAR(100)
);
CREATE TABLE phase2_f (
frontline_nuid VARCHAR(200) PRIMARY KEY,
FOREIGN KEY(frontline_nuid) references frontline_student(frontline_nuid),
eligibility_check VARCHAR(100)
);
CREATE TABLE phase2_n (
non_frontline_nuid VARCHAR(200) PRIMARY KEY,
FOREIGN KEY(non_frontline_nuid) references non_frontline_student(non_frontline_nuid),
eligibility_check VARCHAR(100)
);

CREATE TEMPORARY TABLE T1 (SELECT frontline_nuid FROM frontline_student where
pre_existing = 'YES');

INSERT INTO phase1_f (frontline_nuid, ELIGIBILITY_CHECK)
SELECT frontline_nuid, 'Y' from T1

CREATE TEMPORARY TABLE T1 (SELECT frontline_nuid FROM frontline_student where
pre_existing = 'YES');

INSERT INTO phase1_f (frontline_nuid, ELIGIBILITY_CHECK)
SELECT frontline_nuid, 'Y' from T1

INSERT INTO phase1_r (regular_nuid, eligibility_check)
SELECT regular_nuid, 'Y' from regular_student where pre_existing='YES'

INSERT INTO phase1_n (non_frontline_nuid, eligibility_check)
SELECT non_frontline_nuid, 'Y' from non_frontline_student where pre_existing='YES'

INSERT INTO phase2_n (non_frontline_nuid, eligibility_check)
SELECT non_frontline_nuid, 'Y' from non_frontline_student where pre_existing='NO'

INSERT INTO phase2_r (regular_nuid, eligibility_check)
SELECT regular_nuid, 'Y' from regular_student where pre_existing='NO'
```

```
INSERT INTO phase2_f (frontline_nuid, eligibility_check)
SELECT frontline_nuid, 'Y' from frontline_student where pre_existing='NO'

INSERT INTO phase1_f (frontline_nuid, eligibility_check)
SELECT frontline_nuid,'N' from frontline_student where pre_existing = 'NO'
LIMIT 5
```

Python:

```
import mysql.connector
from password import *
import matplotlib.pyplot as plt
import matplotlib.mlab as mlab
from datetime import date
p=password
mydb = mysql.connector.connect(
  host="localhost",
  user="root",
  password=p,
  database="student"
)

print(mydb)
print('success!!')

query_student_frontline = 'SELECT COUNT(frontline_nuid) from FRONTLINE_STUDENT;'
query_student_regular = 'SELECT COUNT(regular_nuid) from REGULAR_STUDENT;'
query_student_non_frontline = 'SELECT COUNT(non_frontline_nuid) from
NON_FRONTLINE_STUDENT;'

cursor=mydb.cursor()
#mycursor_1 = mydb.cursor()
#mycursor_2= mydb.cursor()
#mycursor_3 = mydb.cursor()

cursor.execute(query_student_frontline)
#mycursor_2.execute("SELECT COUNT(frontline_nuid) FROM REGULAR_STUDENT")
#mycursor_3.execute("SELECT COUNT(frontline_nuid) FROM NON_FRONTLINE_STUDENT")
```

```
for i in cursor:
        number_frontline_student = i[0]
cursor.execute(query_student_regular)
for i in cursor:
        number_regular_student = i[0]
cursor.execute(query_student_non_frontline)
for i in cursor:
        number_non_frontline_student = i[0]
print(number_frontline_student +
number_regular_student +
number_non_frontline_student)

labels = 'Regular', 'Frontline', 'Non Frontline'
sizes = [number_regular_student, number_frontline_student, number_non_frontline_student]

fig1, ax1 = plt.subplots()
ax1.pie(sizes, labels=labels, autopct='%1.1f%%',
    shadow=True, startangle=90)
ax1.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.
ax1.set_title("Distribution of students")
plt.show()

phase1_frontline_student = "SELECT COUNT(FRONTLINE_NUID) from PHASE1_F"
phase1_non_frontline_student = "SELECT COUNT(NON_FRONTLINE_NUID) from PHASE1_N"
phase1_regular_student = "SELECT COUNT(REGULAR_NUID) from PHASE1_R"

cursor.execute(phase1_frontline_student)
#mycursor_2.execute("SELECT COUNT(frontline_nuid) FROM REGULAR_STUDENT")
#mycursor_3.execute("SELECT COUNT(frontline_nuid) FROM NON_FRONTLINE_STUDENT")
for i in cursor:
        number_frontline_student_phase1 = i[0]
cursor.execute(phase1_regular_student)
for i in cursor:
        number_regular_student_phase1 = i[0]
cursor.execute(phase1_non_frontline_student)
for i in cursor:
        number_non_frontline_student_phase1 = i[0]
```

```
total_phase1=number_regular_student_phase1+number_frontline_student_phase1+number_
non_frontline_student_phase1

phase2_frontline_student = "SELECT COUNT(FRONTLINE_NUID) from PHASE2_F"
phase2_non_frontline_student = "SELECT COUNT(NON_FRONTLINE_NUID) from PHASE2_N"
phase2_regular_student = "SELECT COUNT(REGULAR_NUID) from PHASE2_R"

cursor.execute(phase2_frontline_student)
#mycursor_2.execute("SELECT COUNT(frontline_nuid) FROM REGULAR_STUDENT")
#mycursor_3.execute("SELECT COUNT(frontline_nuid) FROM NON_FRONTLINE_STUDENT")
for i in cursor:
        number_frontline_student_phase2 = i[0]
cursor.execute(phase2_regular_student)
for i in cursor:
        number_regular_student_phase2 = i[0]
cursor.execute(phase2_non_frontline_student)
for i in cursor:
        number_non_frontline_student_phase2 = i[0]
total_phase2=number_regular_student_phase2+number_frontline_student_phase2+number_
non_frontline_student_phase2

labels = 'Phase1', 'Phase2'
sizes = [total_phase1,total_phase2]

fig1, ax1 = plt.subplots()
ax1.pie(sizes, labels=labels, autopct='%1.1f%%',
    shadow=True, startangle=90)
ax1.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.
ax1.set_title("Distribution of students in each Phase")
plt.show()

query_frontline_positive='select f.DOB from frontline_student f, testing_f t, results_f r where
f.frontline_nuid=t.frontline_nuid and t.barcode = r.barcode and r.test_result="Yes";'
cursor.execute(query_frontline_positive)
print("testing 1...")
print(cursor)
dob_list=[]
for i in cursor:
```

```
        dob_list=dob_list +[int(i[0][-4:])]
print("testing 2...")
today=date.today()
year = int(str(today)[0:4])
age_list = []
age_list= map(lambda x: year-x, dob_list)
num_bins = 15

n, bins, patches = plt.hist(age_list, num_bins, normed=1, facecolor='blue', alpha=0.5)

plt.xlabel('Age')
plt.ylabel('Numberof people')
plt.title('Distribution of age for Front Line students who test positive')
plt.show()
```