

CS-GY 6083 Principles of Database Systems

Project 1&2

<https://github.com/Monkiia/CovidVaccineAppointment>

Submitted by

Shensheng Chen (sc4932@nyu.edu)

Riyaz Ahmed (ra3360@nyu.edu)

Tables:

provider (pid, name, phone)

provideraddress (pid, street, city, state, zip code, locationX, locationY)

user (ssn, name, age, phone, priorityid)

useraddress (ssn, street, city, state, zip code, locationX, locationY)

login (ssn, username, password)

distance (ssn, pid, distance)

slotblock (slotid, description)

priority_slot(priorityid, slotid)

user_availability (ssn, slotid)

user_travel_limit (id, distance)

provider_travel_limit(pid, distance)

provider_availability(pid, slotid, capacity)

appointment (ssn, slotid, pid, user_accepted, user_canceled,

user_showedup)

baduser(ssn)

cancelledappointment(ssn, slotid, pid, user_accepted, user_canceled,

user_showedup)

notacceptedappointment(ssn, slotid, pid, user_accepted, user_canceled,
user_showedup)

Foreign Keys:

ProviderAddress pid -> Provider pid

ProviderAvailability pid -> Provider pid

Distance pid -> Provider pid

ProviderTravelLimit pid -> Provider pid

Appointment pid -> Provider pid

CanceledAppointment pid -> Provider pid

NotacceptedAppointment pid -> Provider pid

UserAddress ssn -> User ssn

UserAvailability ssn -> User ssn

Distance ssn -> User ssn

UserTravelLimit ssn -> User ssn

Appointment ssn -> User ssn

CancelledAppointment ssn -> User ssn

NotacceptedAppointment ssn -> Provider ssn

Login ssn -> User ssn

Baduser ssn -> User ssn

Priorirtyslot slotid -> slotblock slotid

Appointment slotid -> slotblock slotid

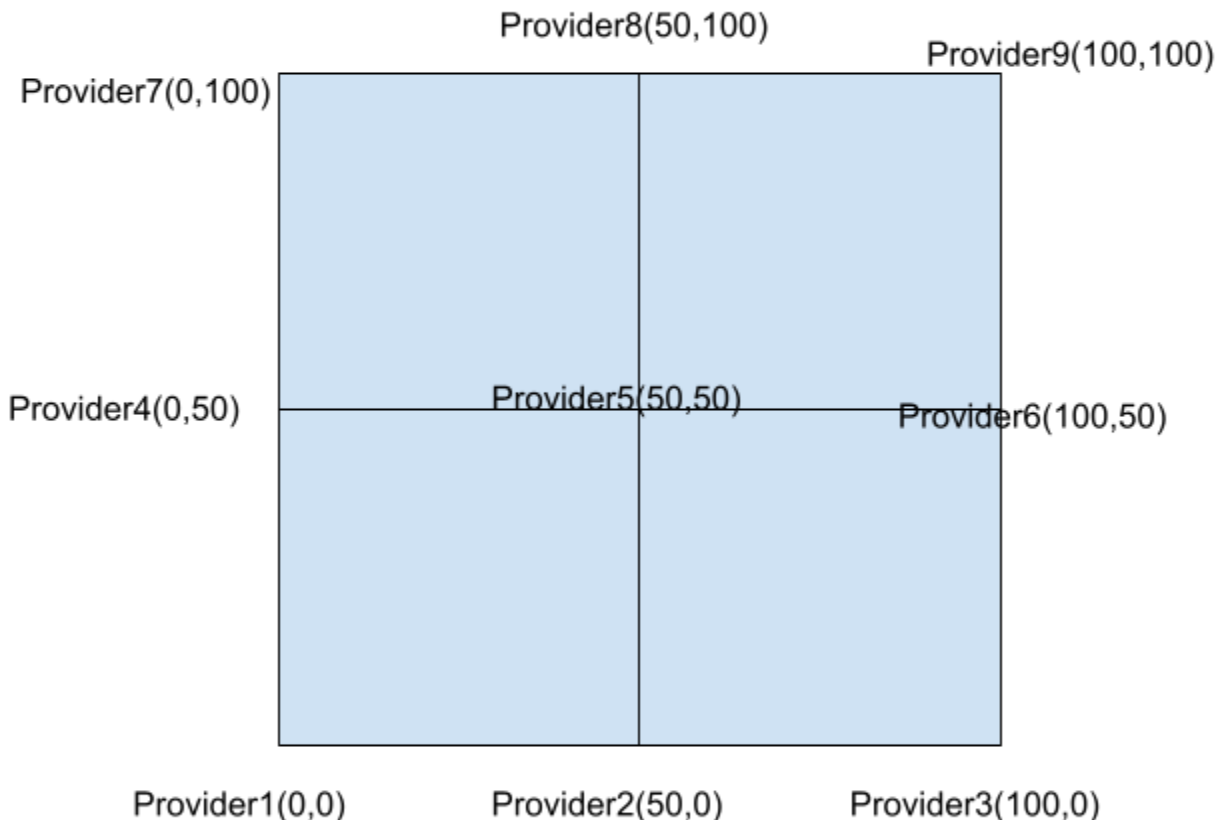
CancelledAppointment slotid -> slotblock slotid

NotacceptedAppointment slotid -> slotblock slotid

Detailed Explanation of table designs:

(for ER diagram, relational graph, SQL Queries are posted after the explanation of our design)

Provider is already in the database. In my design, I have 9 providers who are located in 9 separate symmetric locations.



For testing simplicity, I assign ProviderX zip XXXXX, and Phone XXXXXXXXXXXX

Because we need to the distance between user and provider, I think it's best to quantify them into a coordinate location, (the providers address fixed, and when a user registered, the system will automatically compute his distance to all providers and save it to a table called Distance.

```

4         address.save()
5     f = lambda x1, y1, x2, y2: math.sqrt((x1 - x2) ** 2 + (y1 - y2) ** 2)
6     listproviderinfo = list(Provideraddress.objects.values("pid", "locationx", "locationy"))
7     for i in listproviderinfo:
8         provider_i_pid = i['pid']
9         provider_i_locationx = i['locationx']
10        provider_i_locationy = i['locationy']
11        distancebetweenmeandprovider_i = f(int(LocationX), int(LocationY), int(provider_i_locationx),
12                                           int(provider_i_locationy))
13        distance_me_i = Distance(ssn=SSN, pid=provider_i_pid, distance=distancebetweenmeandprovider_i)
14        distance_me_i.save()

```

I spent some time thinking should I put provider distance limit directly associated with the provider (as a constant) or should I put it when providers add more appointments. It is not flexible if I put it as constant. However, if I associate with each (pid,slotid) a maxdistance, it would be a little bit redundant. So I decided to create a separate provider_travel_limit, every time providers add new appointments, they can also update their travel limit. This is space-efficient without losing too much information.

```

125         return HttpResponse( "You have already added that week schedule ")
126     if verify_provider_already_has_travellimit(providerid):
127         ProviderTravellimit.objects.filter(pid=providerid).delete()
128         new_provider_travel_limit = ProviderTravellimit(pid=providerid, distance=providerdistancelimit)
129         new_provider_travel_limit.save()
130         new_week_appointment = ProviderWeekLock(pid=providerid, week=weekcount)
131         new_week_appointment.save()

```

User, unlike Provider, is not built in, which means we have to implement a user registration service as a function of our project. Our User Registration UI looks like the following:

Personal Information

Please Enter Your name:

Please Enter your Social Security Number:

Please Enter your Age: Format

Please Enter your Phone Number:

Please Enter Your Email Address (This will used for login):

Please Enter Your Password (This will used for login):

Address Information

Please Enter Your Street:

Please Enter Your city:

Please Enter Your State:

Please Enter Your Zipcode:

Please Enter Your LocationX:

Please Enter Your LocationY:

I spent a lot of time thinking about whether or not I should use a random number generated as the userid or should I pick SSN for the primary. I decided to go with SSN because I think that if we generate a random number as a user id, then some sick people will take advantage of our system and register a lot of accounts as bots. While using SSN, I would require the register function to check whether or not there is already an existing identical SSN, if there is, then the user cannot register. (identity theft is out of our consideration because everyone should keep their personal information safe) I also make some checks during registration to make sure that the input is proper instead of something completely wrong or malicious.

In my Django views.touserregisterdatainput, I computed the priorityid given the user's age and save it as a column of user.

```

182     intage = int(Age)
183     prioritygroup = 4
184     if intage >= 70:
185         prioritygroup = 1
186     elif intage >= 55:
187         prioritygroup = 2
188     elif intage >= 40:
189         prioritygroup = 3

```

I also computed distance as I already mentioned.

The **slotblock** table is just a table that converts slotid to time description. The **priorityslot** table is never used in my design, it specifies which priority date the priority group could get vaccines. However, in real-life scenario, (or even in our database design), it is very possible during some time, there might be more provider availability than the corresponding group availability. Should we waste our precious vaccines? No! Thus my way is to order the possibly matched appointment results by first priority ascending then distance ascending.

User_availability table is used for the updated user weekly schedule. If user can log in to the user_availability page, and then what he modified will be saved in this table. However, in our design, the user could only get into this page if he doesn't have any pending appointments to confirm or the user already confirmed his appointment. If so, the user can not change his schedule.

```

51 def touserschedule(request):
52     userssn = request.POST.get('SSN', '')
53     context = {"SSN": userssn}
54     if Appointment.objects.filter(ssn=userssn, user_accepted="pending"):
55         return HttpResponse("You having pending appointment to check. You cannot modify schedule unless you decline your current")
56     if Appointment.objects.filter(ssn=userssn, user_accepted="True"):
57         if Appointment.objects.filter(ssn=userssn, user_accepted="True", user_canceled="pending"):
58             return HttpResponse("You already accepted an offer! No need to modify your schedule")
59     #return HttpResponse(request.POST.get('SSN', ''))
60     return render(request, 'UserSchedule.html', context)

```

Provider_availability is similar, the provider cannot modify/read his appointments for a given week once he specifies that week. (with the corresponding provider_week_lock)

LOCK WEEK Schedule (You can not change it after submit)
ENTER YOUR CAPACITY FOR THIS WEEK
Enter your Maximum Distance Limit

Pick Your available slots

Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
<input type="checkbox"/> Morning	<input type="checkbox"/> Morning	<input type="checkbox"/> Morning	<input type="checkbox"/> Morning	<input type="checkbox"/> Morning	<input type="checkbox"/> Morning	<input type="checkbox"/> Morning
<input type="checkbox"/> Afternoon	<input type="checkbox"/> Afternoon	<input type="checkbox"/> Afternoon	<input type="checkbox"/> Afternoon	<input type="checkbox"/> Afternoon	<input type="checkbox"/> Afternoon	<input type="checkbox"/> Afternoon
<input type="button" value="Submit"/>						

Also, since different providers can have different capacities to hold appointments, we let providers add the capacity by themselves. For example, Kaiser Permanente at Oakland could have more capacity than an individual doctor at San Francisco. Thus during that week, all specified slots will be initialized with that capacity. (I give each (pid, slotid) a corresponding capacity, and I will explain why I do this way instead of creating another table) later.

Appointment

This is the most challenging table because we have so many things to consider. First, let's think about how to get possibly ok appointments, we join **User_availability** and **Provider_availability** with the restriction that the distance should be smaller than both **user_distance_limit** and **provider_distance_limit** then we order our result by **priorityid ascending**, **distance ascending**.

My next step is to iterate through our premade table and check which one fits, if the user is a bad user, then that row is out, if the user has already been picked, then that row is out if the (provider,slot) capacity (one pick decrease one) drops to 0, then after that, all rows with that (provider,slot) will not be considered. By our algorithm, we guaranteed that there will be NO USER HAVING TWO PENDING appointments. After our iteration, we also update provider,slot -> capacity in **Provider_availability** because they could still use it in future rounds.

There is still much more to consider, for example, what if we picked a user but that user already has a pending offer in Appointments (not just in premade data?), we need to rule out that case. What if a user received a pending offer, declined it, but next round the algorithm still picks him?? Maybe the user declined for some reason (other than want to change the schedule) maybe he just doesn't like that specific appointment at that specific timeslot or that specific provider. Our algorithm should not accept him if the previous offering provider still has that slot availability, we should skip that user at this specific time.

See the following code that tackles some of the circumstances:


```
303 slotid = i[5]
304
305
306 # 如果user 已经在真实的appointment 列表中(却还未确认), 跳过
307 if Appointment.objects.filter(ssn=ssn, user_accepted="pending").exists():
308     continue
309 if Appointment.objects.filter(ssn=ssn, user_accepted=True).exists():
310     continue
311 if Cancelledappointment.objects.filter(ssn=ssn, slotid=slotid, pid=provider).exists():
312     continue
313 if (provider_slotid) not in provider_slot_hashmap.keys():
314     if determinebaduser(ssn):
315         continue
316     if ssn in userhashset:
317         continue
318     provider_slot_hashmap[(provider_slotid)] = providercapacity - 1
319     userhashset.append(ssn)
320 # Here to save this particular appointment to our appointment table
321 print("SSN = " + str(ssn))
322 print("provider, slot = " + str((provider_slotid)))
323 print("distance = " + str(distance))
324 newappointment = Appointment(ssn=ssn, slotid=slotid, pid=provider, user_accepted="pending", user_canceled="pending", user_
325 newappointment.save()
326 elif provider_slot_hashmap[(provider_slotid)] == 0:
327     continue
328 else:
329     if ssn in userhashset:
330         continue
331
332 provider_slot_hashmap[(provider_slotid)] = provider_slot_hashmap[(provider_slotid)] - 1
333 inserttimeslot()
334
335 3 files committed, finished bad user and provider see appointment ui
336
337 Pushed 1 commit to origin/main (27 minutes ago) 63:39 CRLF U
```

Badusers

In our design, each time a user cancels or declines, he will be logged at **badusers**, and if we find that he appears more than 3 times in **badusers** He will 'socially dead' and not be able to get access to his page. However, his availability will not be deleted for research purposes. Thus each time we run our algorithm, we have to be careful not to give badusers appointments.

Framework & Language:

We use Python + Django + SQLite for this project because they are very easy to deploy and 'lightweight'. Also it is good for security because Django ORM is safe from SQL injection and all raw SQL we wrote was in static (means no injection). By our asynchronous design of periodic check, we minimize the possibility of any resource contention and concurrency issues. However, there are still some problems with this approach. Doing migrations is painful and Django (up till now) does not support composite primary keys, thus sometimes I have to write additional conditions in python to make sure that I could achieve the result I want.

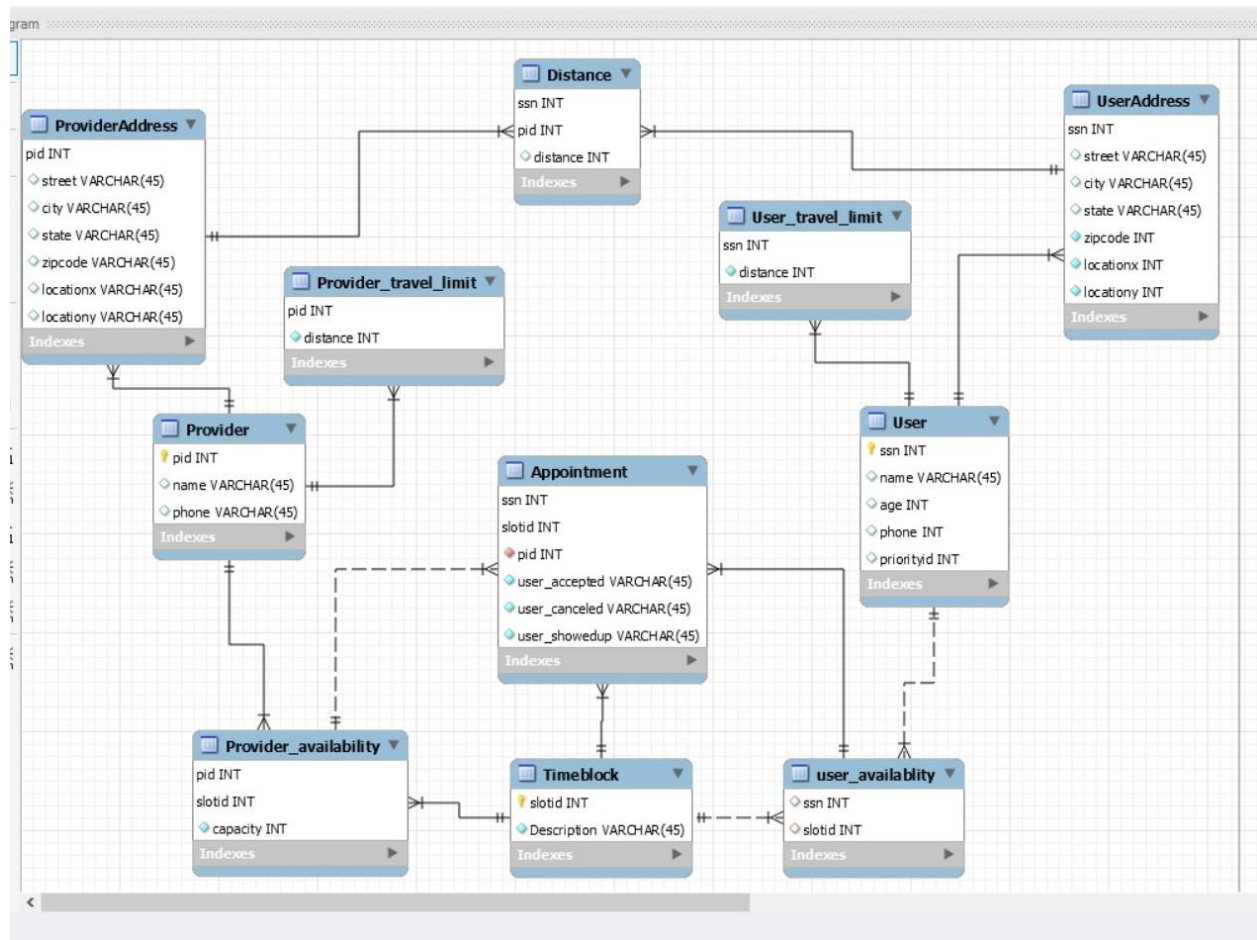
Drawbacks & Things that we need to improve:

One problem with our design is we don't really consider elapsing time, thus we cannot really give anything useful about whether the user showed up or the current time is larger than some slot time thus even it is available you still can't use it. In other words, our project is 'static' which requires more work to make it run into real-life scenario.

Another thing is we provide minimal UI for both users and providers. Yes, the user can see details of provider, and distance, all the functionalities required in the project requirement description, but beyond that there is nothing. The same is the case with the provider. The reason is because of the poor front-end skills, combined with the time factor as well as all the other assignments and projects and exams going on, we were not able to make a dynamic interactive website. (For example, for user to see his past schedule preference)

Other parts that are required in Part1:

EER Diagram:



SQLQueries: Write SQL queries (or sequence of SQL queries or scripting language statements) for the following tasks. You May use suitable placeholder values in the statements (HERE I PUTE CODES AND SCREENSHOT results)

Create a new patient account, together with email,password, name, date of birth, etc.

```
Priorityid = function(Age)
Newpatient = User(ssn = SSN, age = Age, phone = Phone,priotiryid = Priorityid)
Logininfo = Login(ssn = SSN, email = EMAIL, password = PASSWORD)
Newpatient.save()
Logininfo.save()
```

Trivial so I am not uploading the pics

Insert a new appointment offered by a provider.

```
for (provider,slotid) in provider_slot_hashmap.keys():
    print("Provider,slotid " + str((provider,slotid)) + " still have " +
str(provider_slot_hashmap[(provider,slotid)]) + " chances")
    update_query = 'UPDATE provider_availability SET capacity = ? WHERE pid = ?
AND slotid = ?'

cur.execute(update_query, (provider_slot_hashmap[(provider,slotid)],provider,slo
tid))
    print(cur.fetchall())
    print("Executed")
    #provider_availability_object.capacity =
provider_slot_hashmap[(provider,slotid)]
    #provider_availability_object.save()
```

```
con.commit() #Be sure to commit! I spent an hour figuring out what's wrong T
con.close()
```

Trivial so i am not uploading the pics

Write A Query That, forgive patient ssn 357550302, finds available(not currently assigned) appointments that satisfy the constraints on the patient's weekly schedule, sorted by increasing distance from the user's home address

```
431 def toquery3(request):
432     con = sqlite3.connect('db.sqlite3')
433     cur = con.cursor()
434     sql_query = '''SELECT provider.pid, user.priorityid, distance.distance, provider_availability.capacity, user_availability.slotid
435 FROM user, provider, distance, provider_availability, user_availability, user_travel_limit, provider_travel_limit
436 WHERE user.ssn = 357550302
437       AND provider.pid = distance.pid
438       AND provider.pid = provider_availability.pid
439       AND user.ssn = user_availability.ssn
440       AND user_availability.slotid = provider_availability.slotid
441       AND provider_availability.capacity > 0
442       AND user_travel_limit.ssn = user.ssn
443       AND provider_travel_limit.pid = provider.pid
444       AND distance.distance <= user_travel_limit.distance
445       AND distance.distance <= provider_travel_limit.distance
446       AND user.ssn NOT in (SELECT ssn from appointment)
447 ORDER BY user.priorityid ASC, distance.distance ASC
448     '''
449     cur.execute(sql_query)
450     data = cur.fetchall()
```

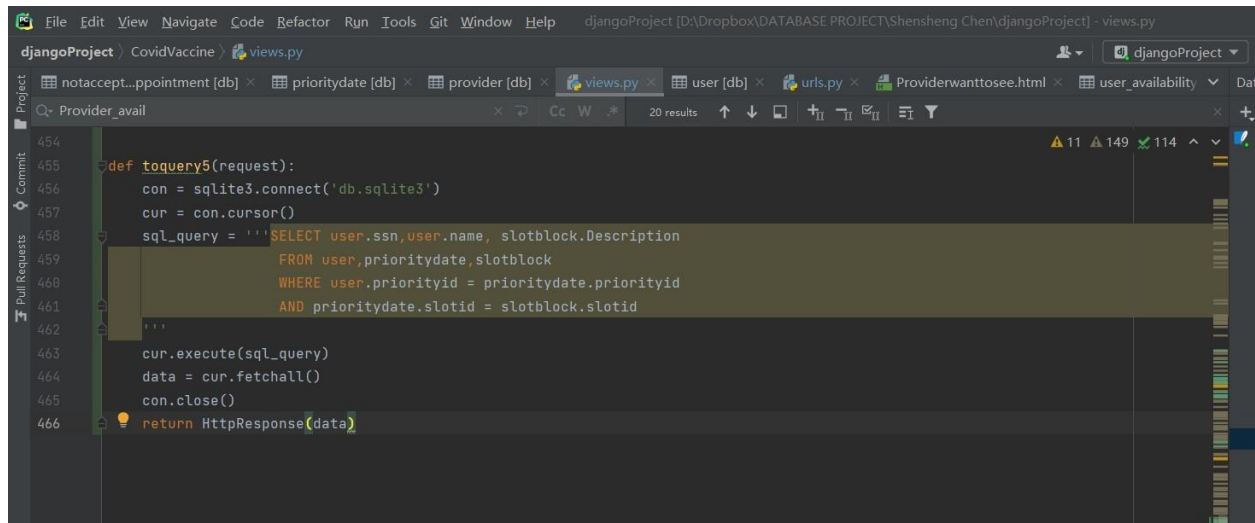
127.0.0.1:8000/CovidVaccine/Query3

(1, 1, 7.0710678118654755, 4, 1)(2, 1, 45.27692569068709, 1, 1)(2, 1, 45.27692569068709, 1, 2)(2, 1, 50.0, 1, 1)(2, 1, 50.0, 1, 2)(2, 1, 105.11898020814318, 1, 1)(2, 1, 105.11898020814318, 1, 2)

For each priority group, list the number of patient that have read received the vaccination, the number of patients currently scheduled for an appointment, and the number of patients still waiting for an appointment.

```
SELECT
Priorityid
From user,appointment
Group by priorityid
```

For each patient, output the ID, name, and date when the patient becomes eligible for vaccination.



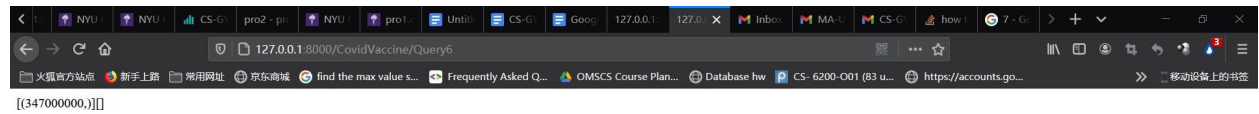
```
454
455 def toquery5(request):
456     con = sqlite3.connect('db.sqlite3')
457     cur = con.cursor()
458     sql_query = '''SELECT user.ssn, user.name, slotblock.Description
459                   FROM user, prioritydate, slotblock
460                   WHERE user.priorityid = prioritydate.priorityid
461                   AND prioritydate.slotid = slotblock.slotid
462                   '''
463     cur.execute(sql_query)
464     data = cur.fetchall()
465     con.close()
466     return HttpResponse(data)
```

127.0.0.1:8000/CovidVaccine/Query5
(357550302, 'Shensheng', 'May 1 8:00 AM - 11:59 AM')(347000000, 'Riyaz', 'August 1 8:00 AM - 11:59 AM')(999999999, 'Zhangsan', 'May 1 8:00 AM - 11:59 AM')

Output the ID and name of all patients that have cancelled at least 3 appointments, or that did not show for at least two confirmed appointments that they did not cancel.

```
File Edit View Navigate Code Refactor Run Tools Git Window Help  djangoProject [D:\Dropbox\DATABASE PROJECT\Shensheng Chen\djangoProject] views.py
djangoProject CovidVaccine > Views.py
notaccept...ppointment [db] x prioritydate [db] x provider [db] x views.py x urls.py x cancelledappointment [db] x user [db] x Providerwanttosee.html x user_availability [db] x baduser [db] x
11 results
464 data = cur.fetchall()
465 con.close()
466 return HttpResponse(data)
467
468 def toquery6(request):
469     con = sqlite3.connect('db.sqlite3')
470     cur = con.cursor()
471     sql_query = '''SELECT ssn
472                 FROM cancelledappointment
473                 group by ssn
474                 having count(*) >= 3
475                 '''
476     cur.execute(sql_query)
477     data1 = cur.fetchall()
478     sql_query = '''SELECT ssn
479                 FROM appointment
480                 WHERE user_showedup = "False"
481                 group by ssn
482                 having count() >= 2
483                 '''
484     cur.execute(sql_query)
485     data2 = cur.fetchall()
486     listofdata = [data1, data2]
487     con.close()
488     return HttpResponse(listofdata)

toquery6()
Pushed 1 commit to origin/main (today 8:08 AM) Python 3.9 (djangoProject) (2) main
```



Output the ID and name of the provider(s) that has performed the largest number of vaccinations.

djangoProject CovidVaccine > views.py

```
482         group_by = 'pid'
483         having count() >= 2
484     cur.execute(sql_query)
485     data2 = cur.fetchall()
486     listofdata = [data1,data2]
487     con.close()
488     return HttpResponse(listofdata)
489
490
491 def toquery7(request):
492     con = sqlite3.connect('db.sqlite3')
493     cur = con.cursor()
494     sql_query = '''SELECT pid, name
495                   FROM provider, appointment
496                   WHERE provider.pid=appointment.pid
497                   AND user_showedup = "True"
498                   GROUP BY pid, name
499                   ORDER BY COUNT(*) DESC
500                   LIMIT 1
501                   '''
502     cur.execute(sql_query)
503     data = cur.fetchall()
504     con.close()
505     return HttpResponse(data)
506
```

Database

- db 1
 - main
 - tables 30
 - appointment
 - ssn int
 - slotid int
 - pid int
 - user_accepted varchar(20)
 - user_canceled varchar(20)
 - user_showedup varchar(20)
 - id INTEGER (auto Increment)
 - appointment_pk (id)
 - auth_group
 - auth_group_permissions
 - auth_permission
 - auth_user
 - auth_user_groups
 - auth_user_user_permissions
 - baduser
 - cancelledappointment
 - distance
 - django_admin_log
 - django_content_type
 - django_migrations
 - django_session
 - notacceptedappointment
 - prioritydate
 - priorityid integer
 - slotid int
 - provider
 - provider_availability
 - provider_travel_limit
 - provider_week_lock
 - provideraddress
 - slotblock

toquery7()

503:26 CRLF UTF-8 4 spaces Python 3.9 (djangoProject) (2) main

127.0.0.1 NYU CS-G pro2 - p NYU pro1 Untitled CS-G Google 127.0.0.1 127.0.0.1 CS-G Inb... Inb... Inb...

127.0.0.1:8000/CovidVaccine/Query7

火狐官方站点 新手上路 常用网址 京东商城 find the max value s... Frequently Asked Q... OMSCS Course Plan... Database hw CS- 6200-001 (83 u... https://accounts.go...

(7, 'Provider7')