Project Report

IFT 458 - PD 6

Spring 2018

Ashley Mendoza

Vincent Li

Dr. Kuitche

April 12, 2018

Introduction

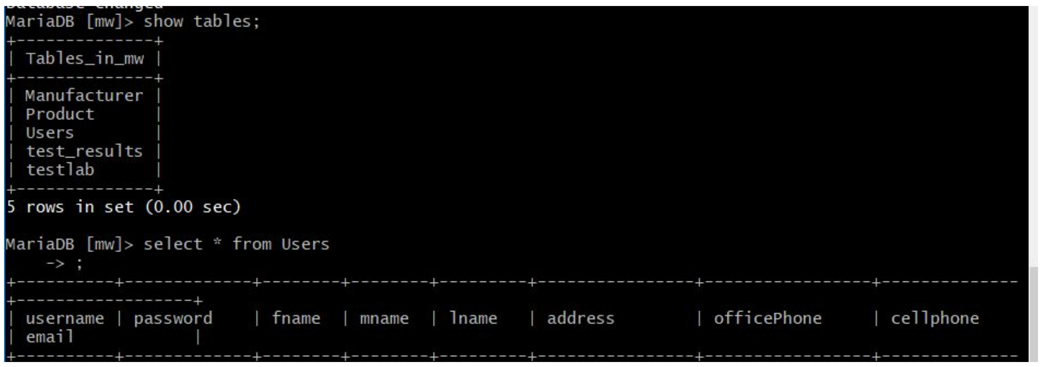
This Project Deliverable will implement the backend created in previous Project Deliverables using Django Framework. The main tasks are to configure the database, create an database application (called backend), create all the necessary tables in the models.py file and use the interactive shell to perform CRUD operations on the database. The following report will outline the steps taken to achieve these tasks and a user manual that guides a user on how to access our project.

Description of Work

This portion of the report will outline the steps taken to accomplish the overall tasks of implementing a fully functioning backend with Django framework. We are assuming here that the database is already created and added to the settings.py file under the “DATABASES” section, shown in Figure 6.0.1 and Figure 6.0.2.

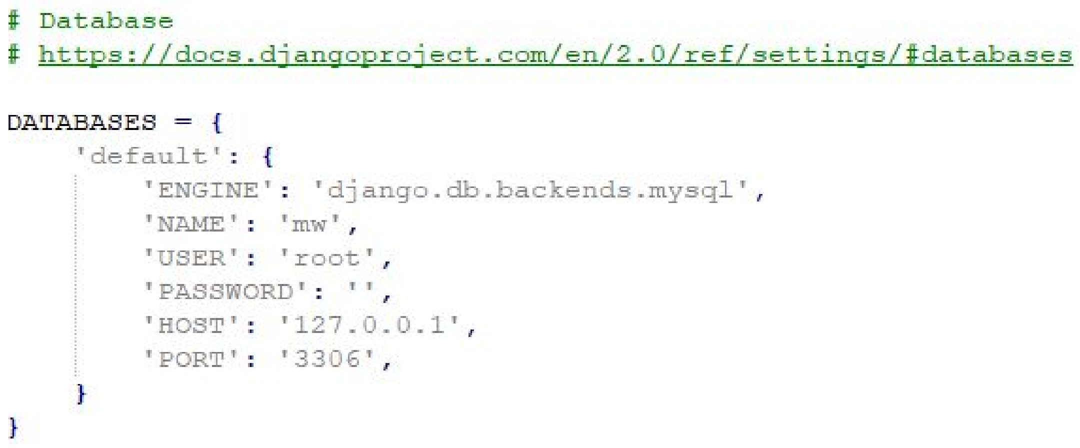
Before running any commands, activate your virtual environment using the command `source myproject_env2/bin/activate`. After, migrate the manage.py file, then use the commands `mysql -u root`, `USE database;`, `SHOW tables;` to verify the database exists, shown in Figure 6.0.3.

Figure 6.0.1: Database already create from previous PD



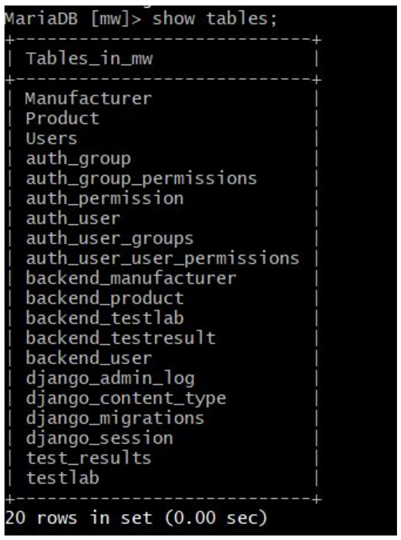
```
MariaDB [mw]> show tables;
+--------------+
| Tables_in_mw |
+--------------+
| Manufacturer |
| Product      |
| Users        |
| test_results |
| testlab      |
+--------------+
5 rows in set (0.00 sec)

MariaDB [mw]> select * from Users
    -> ;
+----------+------------+-------+-------+--------+---------------+----------------+-------------
+-----------------+
| username | password   | fname | mname | lname  | address       | officePhone    | cellphone
| email           |
+----------+------------+-------+-------+--------+---------------+----------------+-------------
```

Figure 6.0.2: Database added in settings.



```
# Database
# https://docs.djangoproject.com/en/2.0/ref/settings/#databases

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'mw',
        'USER': 'root',
        'PASSWORD': '',
        'HOST': '127.0.0.1',
        'PORT': '3306',
    }
}
```

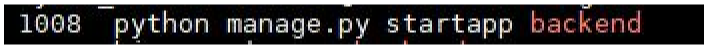
Figure 6.0.3: Database created verification



```
MariaDB [mw]> show tables;
+----------------------------+
| Tables_in_mw               |
+----------------------------+
| Manufacturer               |
| Product                    |
| Users                      |
| auth_group                 |
| auth_group_permissions     |
| auth_permission            |
| auth_user                  |
| auth_user_groups           |
| auth_user_user_permissions |
| backend_manufacturer       |
| backend_product            |
| backend_testlab            |
| backend_testresult         |
| backend_user               |
| django_admin_log           |
| django_content_type        |
| django_migrations          |
| django_session             |
| test_results               |
| testlab                    |
+----------------------------+
20 rows in set (0.00 sec)
```

Figure 6.1 below demonstrates the first step, creating a database application using the command `python manage.py startapp backend`. The startapp allows us to create an app which we names 'backend' accordingly to the instructions.

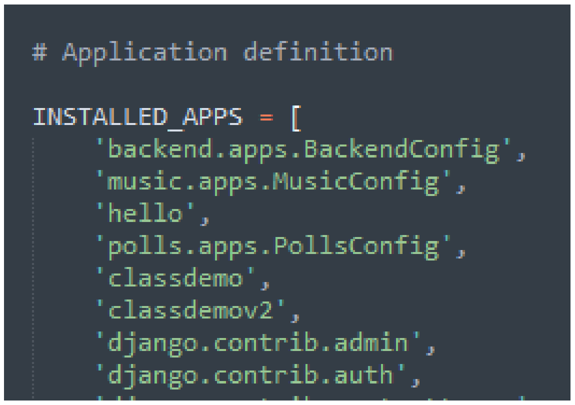
Figure 6.1: Creating new backend app



```
1008  python manage.py startapp backend
```

The next step is to add the newly created app above to the settings.py file under the "INSTALLED_APPS" section. Figure 6.2 demonstrates the edited file with the new app implemented.

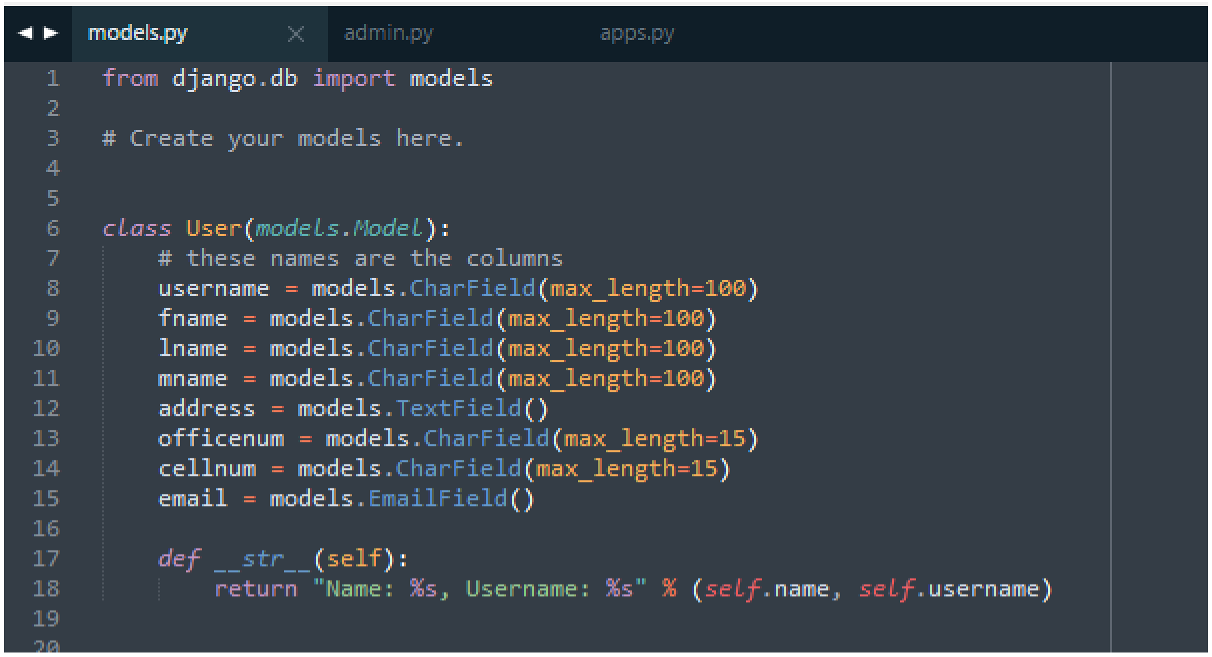
Figure 6.2: Adding backend app to list of installed apps



```
# Application definition

INSTALLED_APPS = [
    'backend.apps.BackendConfig',
    'music.apps.MusicConfig',
    'hello',
    'polls.apps.PollsConfig',
    'classdemo',
    'classdemov2',
    'django.contrib.admin',
    'django.contrib.auth',
```

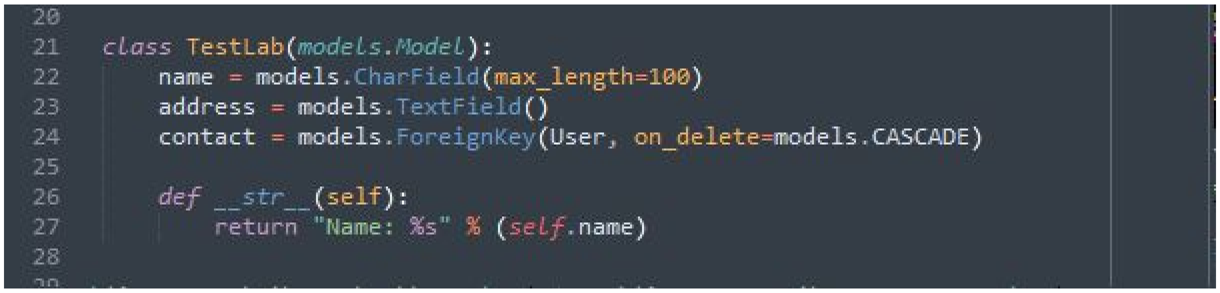
Next, we need to edit the models.py file by adding the classes (creating the tables that are in our backend) and their respective attributes. Here we have different field types depending on the data contained in that field (ie. CharField, TextField, EmailField, DateField). If there is a relationship amongst the table, it will contain a ForeignKey or ManyToMany as the field type. Additionally, the class also returns a string. This string is returned when the backend is queried in the Python terminal. Figure 6.3.1 - Figure 6.3.3 shows the tables implemented in our models.py file.

Figure 6.3.1: Adding classes to backend/models.py



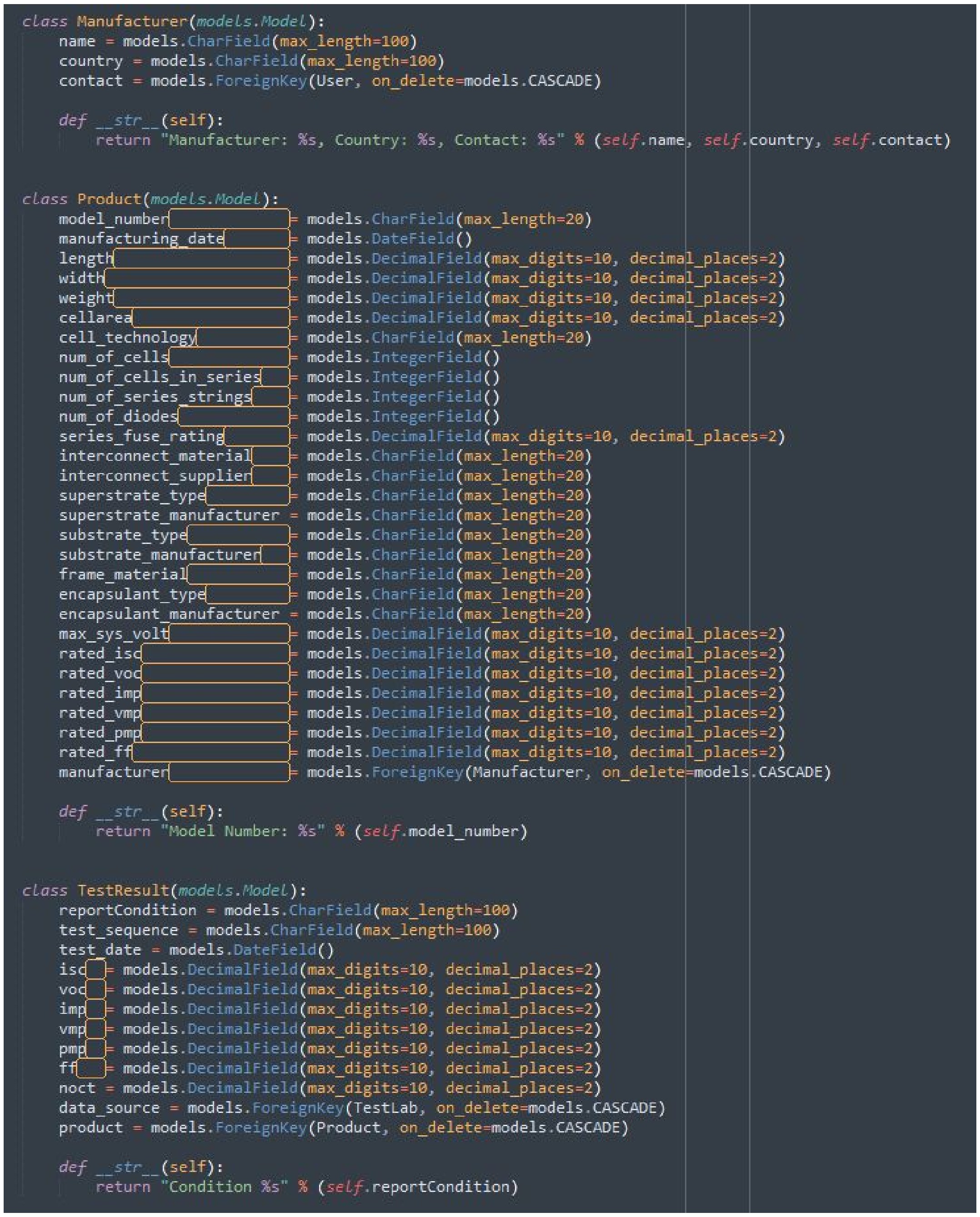
```python
from django.db import models

# Create your models here.


class User(models.Model):
    # these names are the columns
    username = models.CharField(max_length=100)
    fname = models.CharField(max_length=100)
    lname = models.CharField(max_length=100)
    mname = models.CharField(max_length=100)
    address = models.TextField()
    officenum = models.CharField(max_length=15)
    cellnum = models.CharField(max_length=15)
    email = models.EmailField()

    def __str__(self):
        return "Name: %s, Username: %s" % (self.name, self.username)

```

Figure 6.3.2: Adding classes to backend/models.py



```python

class TestLab(models.Model):
    name = models.CharField(max_length=100)
    address = models.TextField()
    contact = models.ForeignKey(User, on_delete=models.CASCADE)

    def __str__(self):
        return "Name: %s" % (self.name)

```

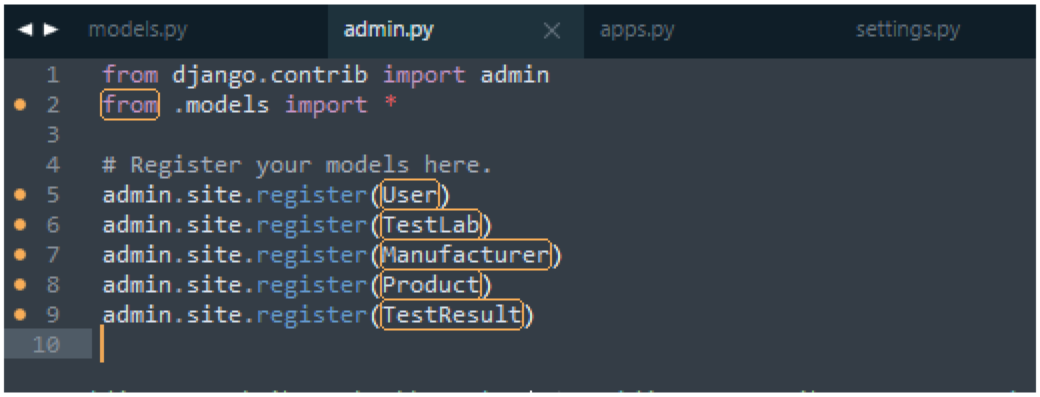
Figure 6.3.3: Adding classes to backend/models.py



```python
class Manufacturer(models.Model):
    name = models.CharField(max_length=100)
    country = models.CharField(max_length=100)
    contact = models.ForeignKey(User, on_delete=models.CASCADE)

    def __str__(self):
        return "Manufacturer: %s, Country: %s, Contact: %s" % (self.name, self.country, self.contact)


class Product(models.Model):
    model_number              = models.CharField(max_length=20)
    manufacturing_date        = models.DateField()
    length                    = models.DecimalField(max_digits=10, decimal_places=2)
    width                     = models.DecimalField(max_digits=10, decimal_places=2)
    weight                    = models.DecimalField(max_digits=10, decimal_places=2)
    cellarea                  = models.DecimalField(max_digits=10, decimal_places=2)
    cell_technology           = models.CharField(max_length=20)
    num_of_cells              = models.IntegerField()
    num_of_cells_in_series    = models.IntegerField()
    num_of_series_strings     = models.IntegerField()
    num_of_diodes             = models.IntegerField()
    series_fuse_rating        = models.DecimalField(max_digits=10, decimal_places=2)
    interconnect_material     = models.CharField(max_length=20)
    interconnect_supplier     = models.CharField(max_length=20)
    superstrate_type          = models.CharField(max_length=20)
    superstrate_manufacturer  = models.CharField(max_length=20)
    substrate_type            = models.CharField(max_length=20)
    substrate_manufacturer    = models.CharField(max_length=20)
    frame_material            = models.CharField(max_length=20)
    encapsulant_type          = models.CharField(max_length=20)
    encapsulant_manufacturer  = models.CharField(max_length=20)
    max_sys_volt              = models.DecimalField(max_digits=10, decimal_places=2)
    rated_isc                 = models.DecimalField(max_digits=10, decimal_places=2)
    rated_voc                 = models.DecimalField(max_digits=10, decimal_places=2)
    rated_imp                 = models.DecimalField(max_digits=10, decimal_places=2)
    rated_vmp                 = models.DecimalField(max_digits=10, decimal_places=2)
    rated_pmp                 = models.DecimalField(max_digits=10, decimal_places=2)
    rated_ff                  = models.DecimalField(max_digits=10, decimal_places=2)
    manufacturer              = models.ForeignKey(Manufacturer, on_delete=models.CASCADE)

    def __str__(self):
        return "Model Number: %s" % (self.model_number)


class TestResult(models.Model):
    reportCondition = models.CharField(max_length=100)
    test_sequence = models.CharField(max_length=100)
    test_date = models.DateField()
    isc = models.DecimalField(max_digits=10, decimal_places=2)
    voc = models.DecimalField(max_digits=10, decimal_places=2)
    imp = models.DecimalField(max_digits=10, decimal_places=2)
    vmp = models.DecimalField(max_digits=10, decimal_places=2)
    pmp = models.DecimalField(max_digits=10, decimal_places=2)
    ff  = models.DecimalField(max_digits=10, decimal_places=2)
    noct = models.DecimalField(max_digits=10, decimal_places=2)
    data_source = models.ForeignKey(TestLab, on_delete=models.CASCADE)
    product = models.ForeignKey(Product, on_delete=models.CASCADE)

    def __str__(self):
        return "Condition %s" % (self.reportCondition)
```

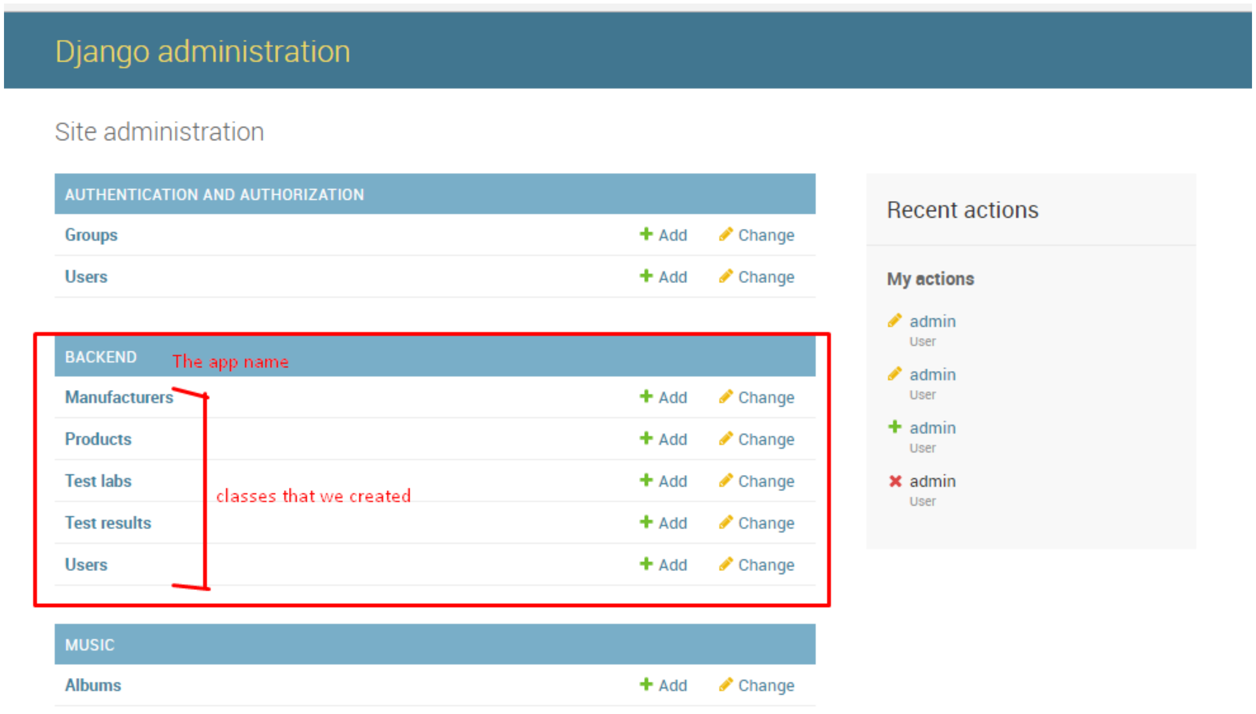
Next, edit the admin.py file to contain access to the tables and create a connection to the model.py file. Figure 6.4 shows the edited file. Figure 6.5 shows the view of the backend in the Django administration GUI form.

Figure 6.4: Adding administrative links



```
from django.contrib import admin
from .models import *

# Register your models here.
admin.site.register(User)
admin.site.register(TestLab)
admin.site.register(Manufacturer)
admin.site.register(Product)
admin.site.register(TestResult)
```

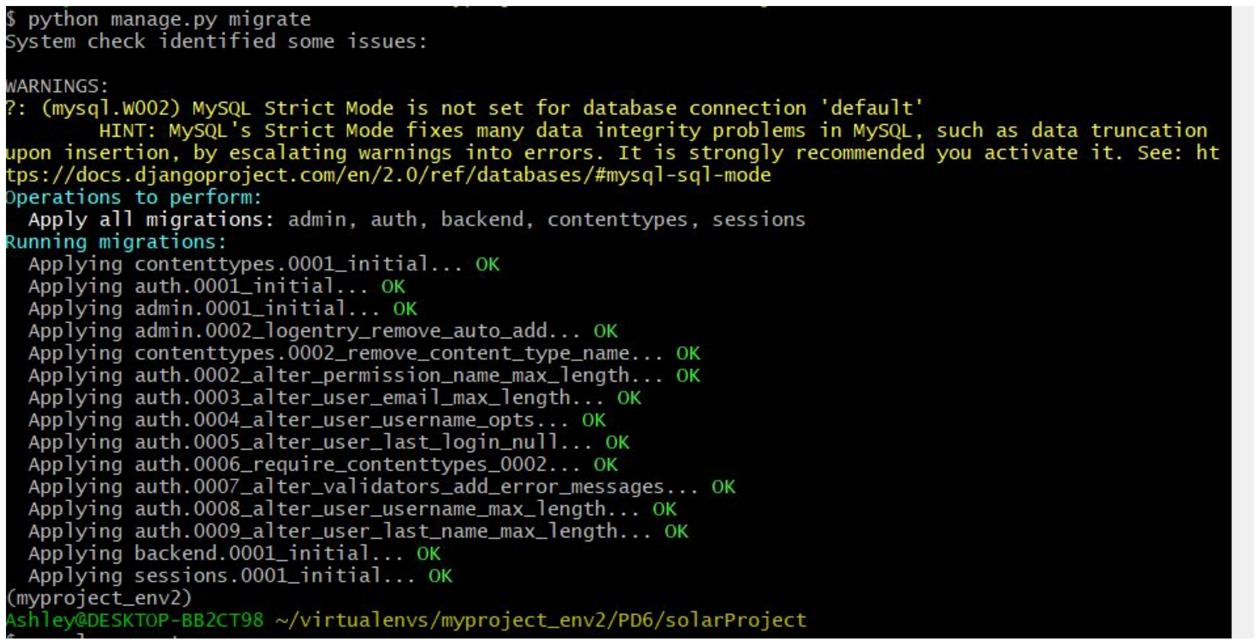
Figure 6.5: View of the admin screen



Now that the backend is set-up and connected, we can now test the functionality of this connection by manipulating the data in the interactive shell by performing CRUD (insert data, retrieve data, update data, and delete data) operations on the database.

Before accessing the python database, we want to ensure that the changes made in our .py files are updated. We do so by running the commands `python manage.py makemigrations` and `python manage.py migrate`. Shown in Figure 6.6.0 below.

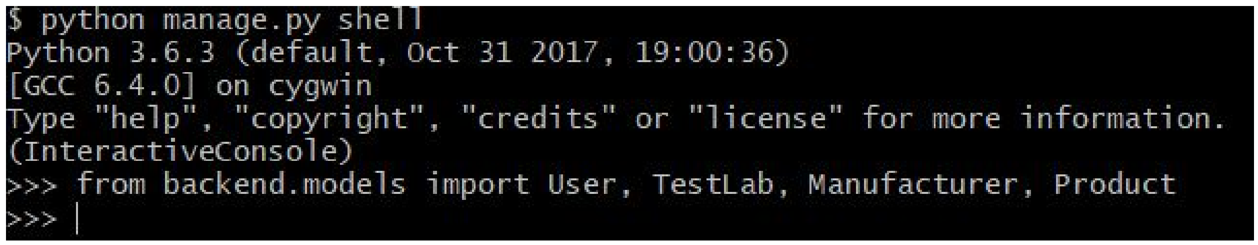
Figure 6.6.0 : Migrating Changes



```
$ python manage.py migrate
System check identified some issues:

WARNINGS:
?: (mysql.W002) MySQL Strict Mode is not set for database connection 'default'
        HINT: MySQL's Strict Mode fixes many data integrity problems in MySQL, such as data truncation
upon insertion, by escalating warnings into errors. It is strongly recommended you activate it. See: ht
tps://docs.djangoproject.com/en/2.0/ref/databases/#mysql-sql-mode
Operations to perform:
  Apply all migrations: admin, auth, backend, contenttypes, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying backend.0001_initial... OK
  Applying sessions.0001_initial... OK
(myproject_env2)
Ashley@DESKTOP-BB2CT98 ~/virtualenvs/myproject_env2/PD6/solarProject
```

Next, we have to access the python interpreter to run the CRUD commands. To access the shell, run the command `python manage.py shell` demonstrated in Figure 6.6.1. Then, import the tables using the command `from backend.models import User, TestLab, Manufacturer, Product, TestResult`.

Figure 6.6.1 : Accessing Interactive Shell/Importing Classes



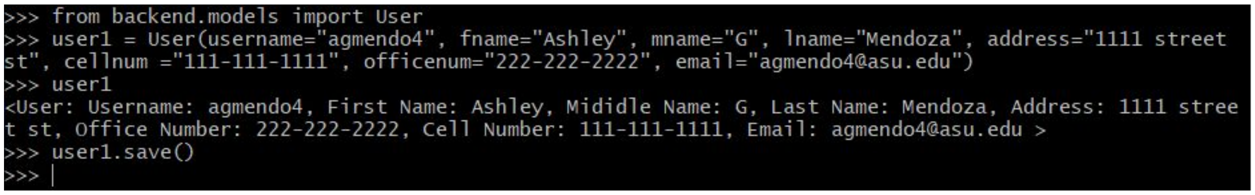
```
$ python manage.py shell
Python 3.6.3 (default, Oct 31 2017, 19:00:36)
[GCC 6.4.0] on cygwin
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>> from backend.models import User, TestLab, Manufacturer, Product
>>> |
```

The first CRUD operation is demonstrated in Figure 6.6.2. Here, we are simply inserting data into the database. This is accomplished by using the command user1 = User(username =

“agmendo4”, fname = “Ashley”, mname=”G”, lname = “Mendoza”, address=”1111 Street st”, cellnum= “222-222-2222, ”officenum = “111-111-1111” email = ”agmendo4@asu.edu”), shown in Figure 6.1. To ensure this is populated in the table, we run the command `user1.save()` to save the row. Repeat these steps to create more users.

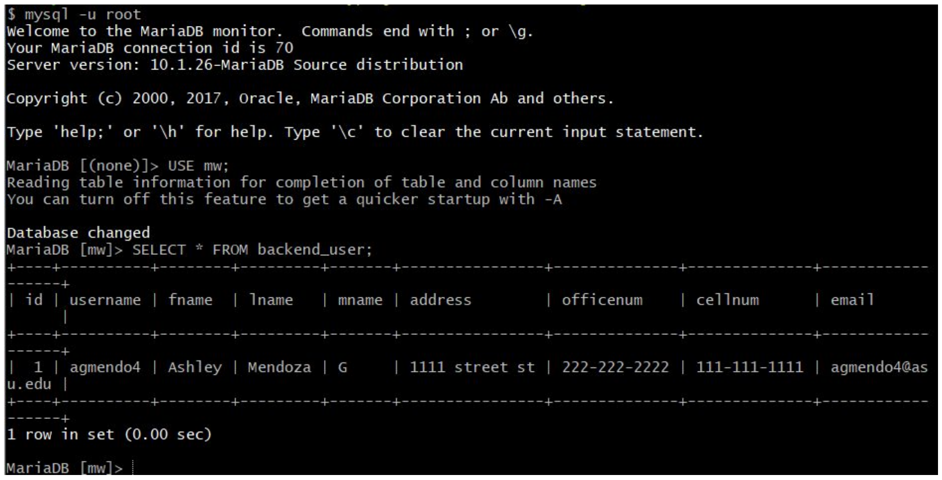
Note: To verify the table was populated, exit the shell interpreter and run the commands: `mysql -u root`, `USE database;`, `SHOW tables;`, `SELECT * FROM backend_User;`, shown in Figure 6.6.3.

Figure 6.6.2: Inserting data into database



```
>>> from backend.models import User
>>> user1 = User(username="agmendo4", fname="Ashley", mname="G", lname="Mendoza", address="1111 street
st", cellnum ="111-111-1111", officenum="222-222-2222", email="agmendo4@asu.edu")
>>> user1
<User: Username: agmendo4, First Name: Ashley, Mididle Name: G, Last Name: Mendoza, Address: 1111 stree
t st, Office Number: 222-222-2222, Cell Number: 111-111-1111, Email: agmendo4@asu.edu >
>>> user1.save()
>>> |
```

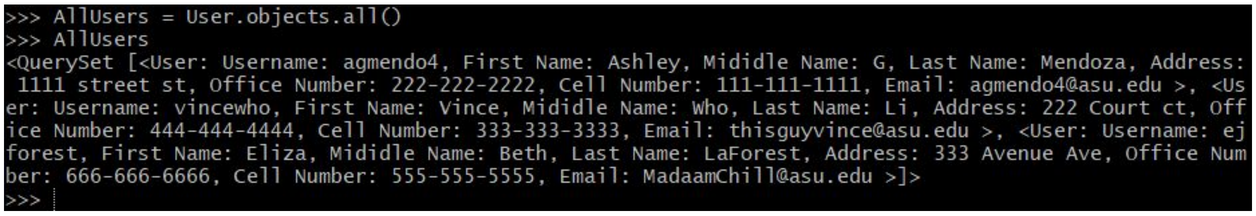
Figure 6.6.3: Populated User Table



```
$ mysql -u root
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 70
Server version: 10.1.26-MariaDB Source distribution

Copyright (c) 2000, 2017, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> USE mw;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
MariaDB [mw]> SELECT * FROM backend_user;
+----+----------+--------+---------+-------+----------------+--------------+--------------+------------
------+
| id | username | fname  | lname   | mname | address        | officenum    | cellnum      | email
      |
+----+----------+--------+---------+-------+----------------+--------------+--------------+------------
------+
|  1 | agmendo4 | Ashley | Mendoza | G     | 1111 street st | 222-222-2222 | 111-111-1111 | agmendo4@as
u.edu |
+----+----------+--------+---------+-------+----------------+--------------+--------------+------------
------+
1 row in set (0.00 sec)

MariaDB [mw]> |
```

Before retrieving data, as shown in Figure 6.7.1, use the commands `AllUsers = User.objects.all()` and `AllUsers` to display all the users that are populated in the database to ensure correct data is being retrieved. As shown in Figure 6.7.1, we have three Users in our table that we will be manipulating for demonstration purposes.

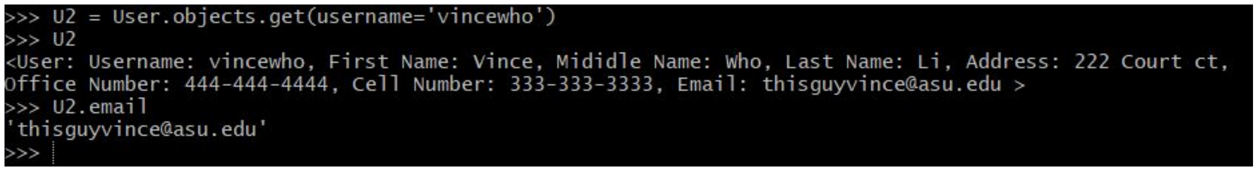
Figure 6.7.1: Retrieving data: Query - All Users in User DB



```
>>> AllUsers = User.objects.all()
>>> AllUsers
<QuerySet [<User: Username: agmendo4, First Name: Ashley, Mididle Name: G, Last Name: Mendoza, Address:
 1111 street st, Office Number: 222-222-2222, Cell Number: 111-111-1111, Email: agmendo4@asu.edu >, <Us
er: Username: vincewho, First Name: Vince, Mididle Name: Who, Last Name: Li, Address: 222 Court ct, Off
ice Number: 444-444-4444, Cell Number: 333-333-3333, Email: thisguyvince@asu.edu >, <User: Username: ej
forest, First Name: Eliza, Mididle Name: Beth, Last Name: LaForest, Address: 333 Avenue Ave, Office Num
ber: 666-666-6666, Cell Number: 555-555-5555, Email: MadaamChill@asu.edu >]>
>>>
```

The second CRUD operation is demonstrated in Figure 6.7.2. Here, we are simply retrieving data from the database. First, query the database and save the returned data into a variable, we can then either display the entire data string or a specific attribute/field of the data object. For example, Figure 6.7.2 demonstrates retrieving object data (row) from database that contains the username: ’vincewho’, and saves it into a variable called U2. Then it displays the value of the U2’s email.

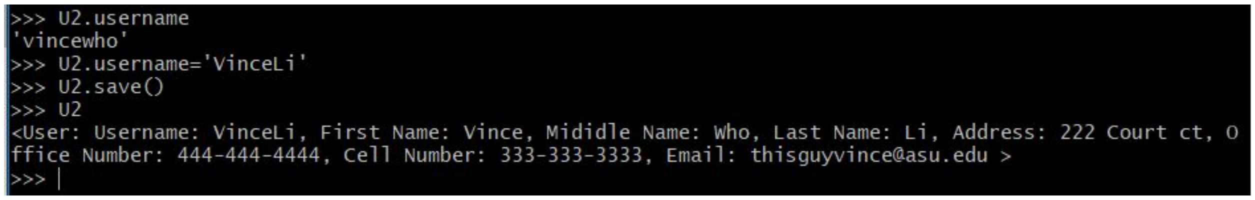
Figure 6.7.2: Retrieving data: Query - User2



```
>>> U2 = User.objects.get(username='vincewho')
>>> U2
<User: Username: vincewho, First Name: Vince, Mididle Name: Who, Last Name: Li, Address: 222 Court ct,
Office Number: 444-444-4444, Cell Number: 333-333-3333, Email: thisguyvince@asu.edu >
>>> U2.email
'thisguyvince@asu.edu'
>>>
```

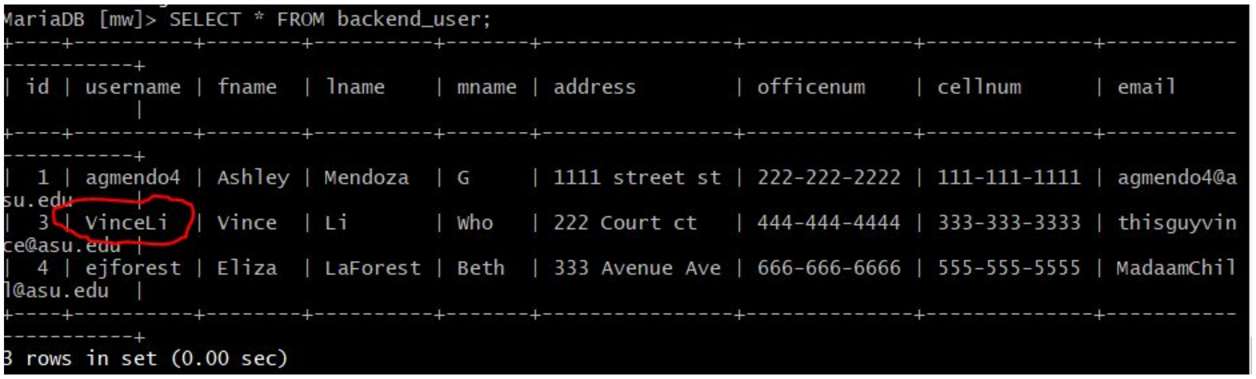
The third CRUD operation is demonstrated in Figure 6.8.1. Here, we are simply updating the data from the database. This is accomplished by referencing the variable containing the object (U2 above), then the attribute and assign it a new value. To ensure the update is finalized, save the object. To verify success, , exit the shell interpreter and run the commands: \`mysql -u root\`, \`USE database;\`, \`SHOW tables;\`, \`SELECT * FROM backend_User;\`, shown in Figure 6.8.2.

Figure 6.8.1: Update Data



```
>>> U2.username
'vincewho'
>>> U2.username='VinceLi'
>>> U2.save()
>>> U2
<User: Username: VinceLi, First Name: Vince, Mididle Name: Who, Last Name: Li, Address: 222 Court ct, O
ffice Number: 444-444-4444, Cell Number: 333-333-3333, Email: thisguyvince@asu.edu >
>>> |
```

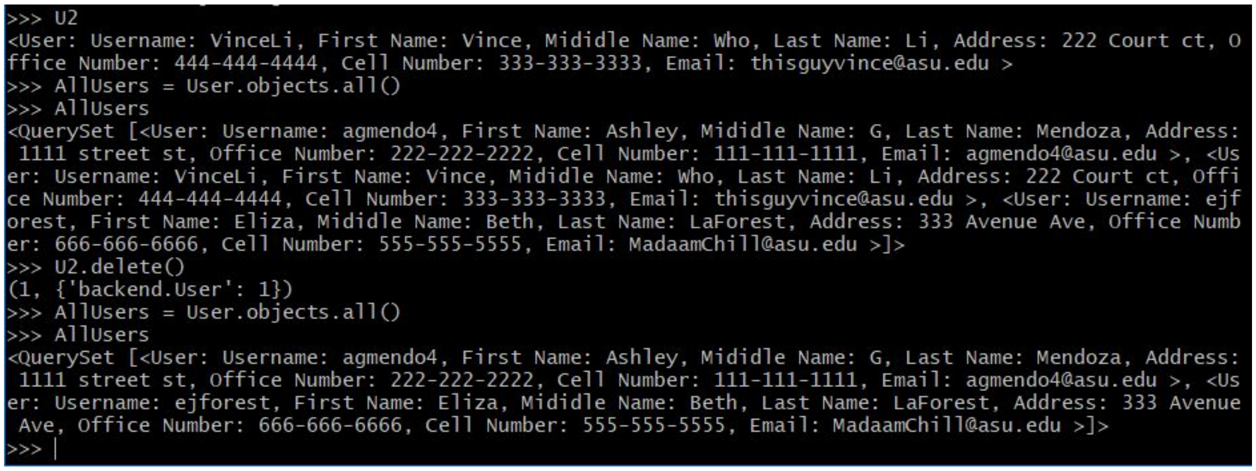
Figure 6.8.2: Update Verification



```
MariaDB [mw]> SELECT * FROM backend_user;
+----+----------+--------+----------+-------+----------------+--------------+--------------+----------
-----------+
| id | username | fname  | lname    | mname | address        | officenum    | cellnum      | email
           |
+----+----------+--------+----------+-------+----------------+--------------+--------------+----------
-----------+
|  1 | agmendo4 | Ashley | Mendoza  | G     | 1111 street st | 222-222-2222 | 111-111-1111 | agmendo4@a
su.edu     |
|  3 | VinceLi  | Vince  | Li       | Who   | 222 Court ct   | 444-444-4444 | 333-333-3333 | thisguyvin
ce@asu.edu |
|  4 | ejforest | Eliza  | LaForest | Beth  | 333 Avenue Ave | 666-666-6666 | 555-555-5555 | MadaamChil
l@asu.edu  |
+----+----------+--------+----------+-------+----------------+--------------+--------------+----------
-----------+
3 rows in set (0.00 sec)
```

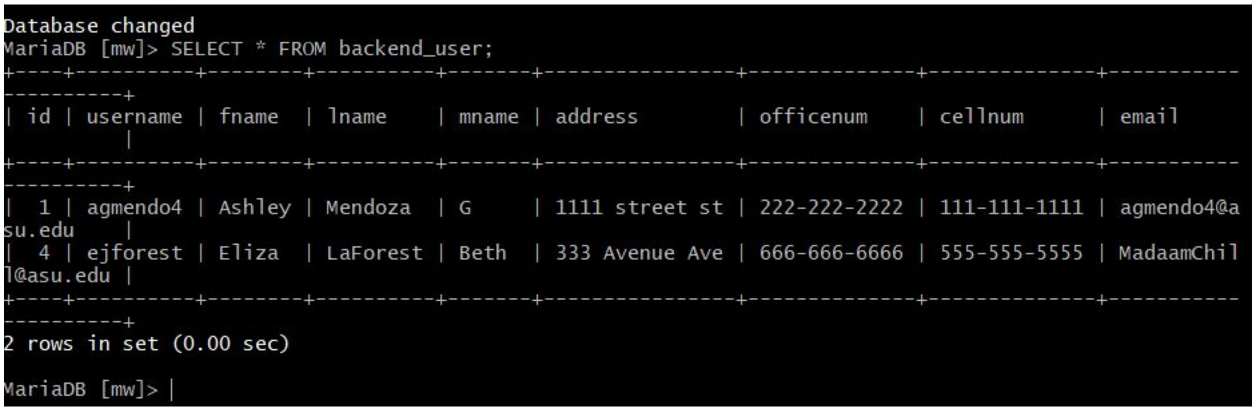
The fourth CRUD operation is demonstrated in Figure 6.9.1. Here, we are simply deleting a data row from the database. This will be done by using the delete method. Using the object variable created above(U2) and the delete method, we will delete the user VinceLi. To verify success, either display the variable AllUsers or exit the shell interpreter and run the commands: `mysql -u root`, `USE database;`, `SHOW tables;`, `SELECT * FROM backend_User;`, shown in Figure 6.9.2.

Figure 6.9.1: Deleting data from database



```
>>> U2
<User: Username: VinceLi, First Name: Vince, Mididle Name: Who, Last Name: Li, Address: 222 Court ct, O
ffice Number: 444-444-4444, Cell Number: 333-333-3333, Email: thisguyvince@asu.edu >
>>> AllUsers = User.objects.all()
>>> AllUsers
<QuerySet [<User: Username: agmendo4, First Name: Ashley, Mididle Name: G, Last Name: Mendoza, Address:
 1111 street st, Office Number: 222-222-2222, Cell Number: 111-111-1111, Email: agmendo4@asu.edu >, <Us
er: Username: VinceLi, First Name: Vince, Mididle Name: Who, Last Name: Li, Address: 222 Court ct, Offi
ce Number: 444-444-4444, Cell Number: 333-333-3333, Email: thisguyvince@asu.edu >, <User: Username: ejf
orest, First Name: Eliza, Mididle Name: Beth, Last Name: LaForest, Address: 333 Avenue Ave, Office Numb
er: 666-666-6666, Cell Number: 555-555-5555, Email: MadaamChill@asu.edu >]>
>>> U2.delete()
(1, {'backend.User': 1})
>>> AllUsers = User.objects.all()
>>> AllUsers
<QuerySet [<User: Username: agmendo4, First Name: Ashley, Mididle Name: G, Last Name: Mendoza, Address:
 1111 street st, Office Number: 222-222-2222, Cell Number: 111-111-1111, Email: agmendo4@asu.edu >, <Us
er: Username: ejforest, First Name: Eliza, Mididle Name: Beth, Last Name: LaForest, Address: 333 Avenue
 Ave, Office Number: 666-666-6666, Cell Number: 555-555-5555, Email: MadaamChill@asu.edu >]>
>>> |
```

Figure 6.9.2: Deletion Verification

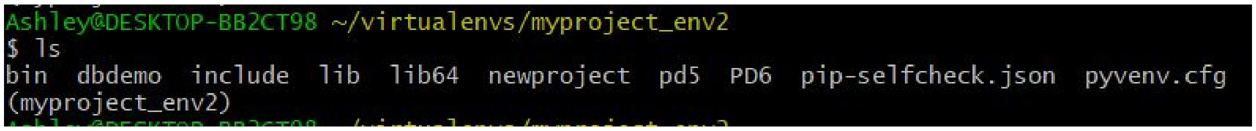


```
Database changed
MariaDB [mw]> SELECT * FROM backend_user;
+----+----------+--------+----------+-------+----------------+--------------+--------------+-----------
----------+
| id | username | fname  | lname    | mname | address        | officenum    | cellnum      | email
          |
+----+----------+--------+----------+-------+----------------+--------------+--------------+-----------
----------+
|  1 | agmendo4 | Ashley | Mendoza  | G     | 1111 street st | 222-222-2222 | 111-111-1111 | agmendo4@a
su.edu    |
|  4 | ejforest | Eliza  | LaForest | Beth  | 333 Avenue Ave | 666-666-6666 | 555-555-5555 | MadaamChil
l@asu.edu |
+----+----------+--------+----------+-------+----------------+--------------+--------------+-----------
----------+
2 rows in set (0.00 sec)

MariaDB [mw]> |
```

User Manual

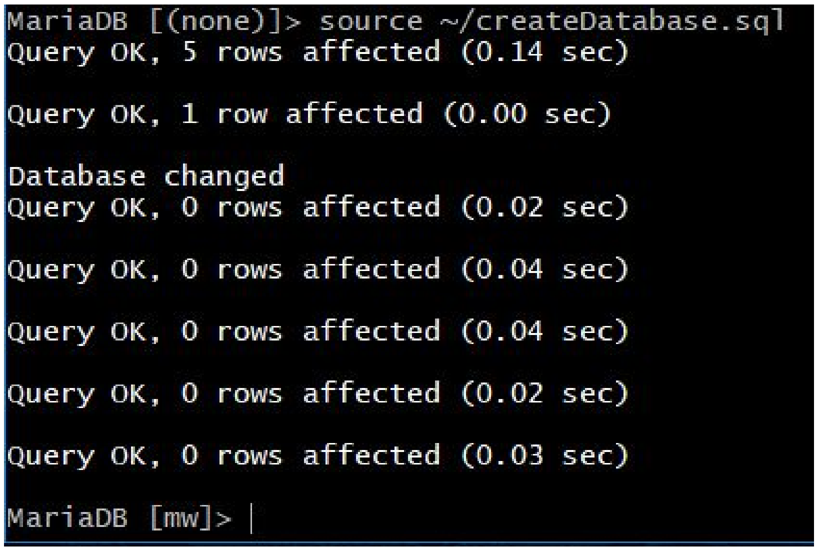
To execute these scripts, ensure that the .zip file provided is downloaded and all files are kept in the same format. Put these scripts under the virtual environment directory (into the folder). Ensure these scripts are present by executing the command `/virtualenv/myproject_env2 ls` shown in Figure 6.10.1 (our file is called PD6). Place the createtables.sql script in your home directory. Login to your database environment, then execute the createtables.sql script to generate the database using the command `source ~/createtables.sql` (Reference Figure 6.10.2). Next, activate your virtual environment using the command `source myproject_env2/bin/activate`. Modify the settings.py file to personal db settings (ex. add a password if necessary). Next migrate the manage.py files to generate the ‘backend’ tables (Reference Figure 6.0.3). Now, you can log into the python interactive shell and perform CRUD statements or filter the database and it will be successfully return data. You may also login to MariaDB and use database/show tables and use other sql statements to query the database.

Figure 6.10.1: Files verification



```
Ashley@DESKTOP-BB2CT98 ~/virtualenvs/myproject_env2
$ ls
bin  dbdemo  include  lib  lib64  newproject  pd5  PD6  pip-selfcheck.json  pyvenv.cfg
(myproject_env2)
```

Figure 6.10.2: CreateTables.sql execution



```
MariaDB [(none)]> source ~/createDatabase.sql
Query OK, 5 rows affected (0.14 sec)

Query OK, 1 row affected (0.00 sec)

Database changed
Query OK, 0 rows affected (0.02 sec)

Query OK, 0 rows affected (0.04 sec)

Query OK, 0 rows affected (0.04 sec)

Query OK, 0 rows affected (0.02 sec)

Query OK, 0 rows affected (0.03 sec)

MariaDB [mw]> |
```

Conclusion

In conclusion, completing this portion of the project allowed us to apply the new concepts learned in class and IFT 433 by implementing our back end into the Django environment. We learned how to create classes(tables), connect to a database and manipulate the data using CRUD statements in the python interactive shell. The main hurdle of this project was learning the proper syntax. I believe this PD was the strongest understanding of concept we've had as a team. Consequently, I do not believe this Project deliverable could have been improved. Overall, this project was useful in grasping a better understanding and reinforce database concepts and how to work in the Django environment.