Data Querying

Let's start this guide by adding new data into the Projects table we finished in the last step:

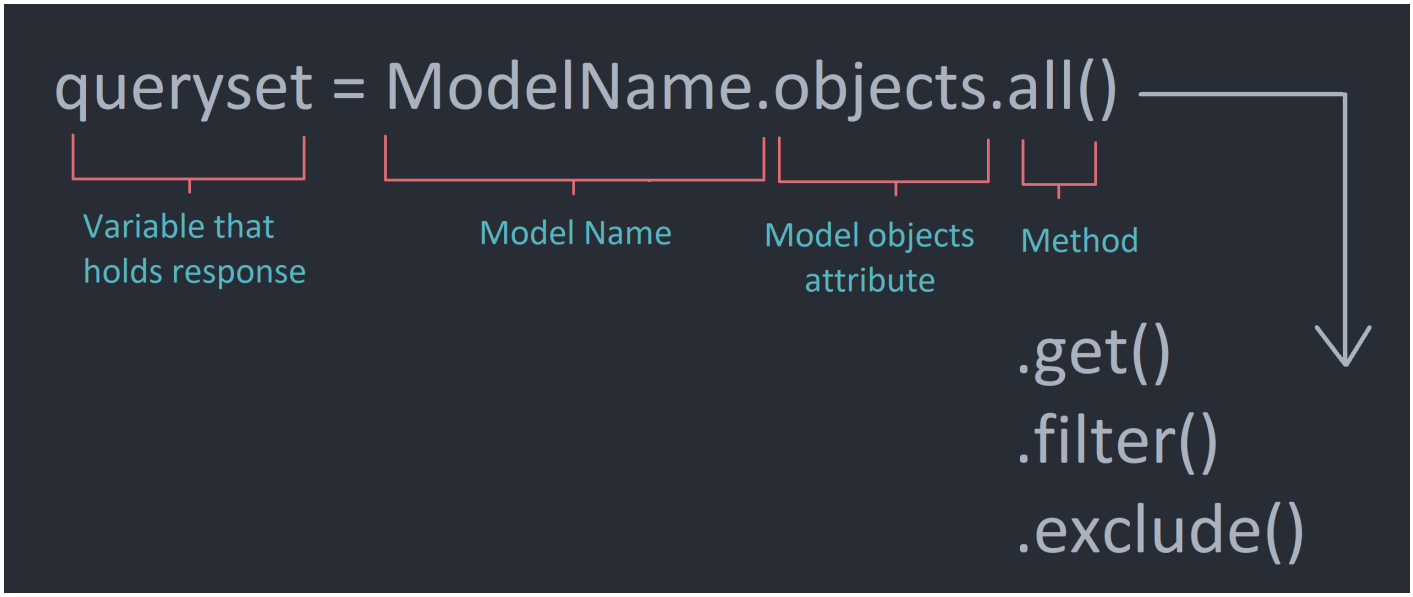
1. You should have at least three projects:
    - Ecommerce Website
    - Portfolio Website
    - Mumble Social Network
2. For each project, add the values of vote total and vote ratio:
    - Ecommerce Website vote total: 12 vote ratio: 30
    - Portfolio Website vote total: 6 vote ratio: 65
    - Mumble Social Network vote total: 89 vote ratio: 74
3. Add two more projects:
    - Code Sniper vote total: 80 vote ratio: 78
    - Yogo Vive vote total: 51 vote ratio: 98

We'll use these data in order to generate queries to consult the database.

Querysets

A query set is, in simple terms, a list of objects (each object is a single record that complies with the conditions imposed on the query).

The following is the model for a queryset



```
queryset - variable holding a reference to the list of records included in
the response to the query to the database

ModelName - is the name of the model (table) for which the query was
targeted.

objects - is a Model.objects attribute (a member property for the Model
object).

all(), get(), filter(), exclude() - are the chained query filter methods.
```

The number of records and the record selection (what records are brought forward or excluded) are determined by the selected method and the arguments used in these chained methods.

IMPORTANT: On normal use, the queryset are located on the <app> / views.py file. However, it doesn't mean that they can't not be used somewhere else either in the application or project.

In general, this is what the queryset method do individually:

1. all(): Brings up all records in the table regardless.
2. get(*attribute='value'*): Brings up the first single record that matches the *attribute == 'value'* with the selection parameter provided to the function.
3. filter(*attribute='value'*): Brings up all the records that match the *attribute == 'value'*. However, here the parameter evaluation is a bit ackward. There are no logical operators involved, so the syntax is straight forward but weird.

   attribute='value' is attribute == 'value' *attribute__startswith='value'* is similar to str.startswith() *attribute__contains='value'* is value in attribute? *attribute__icontains='value'* is value in attribute (case independent)? *attribute__gt='value'* is attribute > 'value' *attribute__gte='value'* is attribute >= 'value' *attribute__lt='value'* is attribute < 'value' *attribute__lte='value'* is attribute <= 'value'
4. exclude(*attribute='value'*) is equivalent to not filter(*attribute='value'*). Exclude any matching record from the query
5. order_by('value1', 'value2', ...) is a sorting modifier for the query set. Usually, this is the last element in the responsability chain (multiple methods ordered and separated by '.' periods.) Multiple values are allowed. The basic idea is to sort the matching records depending on specific values for attributes.
6. create(*attribute='value'*) Creates an instance of a model. It can contain multiple attributes depending on how many attributes are required in order to create a record.
7. save() Saves a recently created record into the Model table.

8. delete Deletes the record. Usually the last element in the responsability chain after selecting a single or multiple records to be deleted.
9. *dependantrelationship_set.all()* ManytoOne access relationship
10. *relationshipname.all()* to query relationships (ManytoMany)
11. *relationshipname.add(otheritem)* to associate records to a ManytoMany relationship

The examples of all these methods are shown in the following:

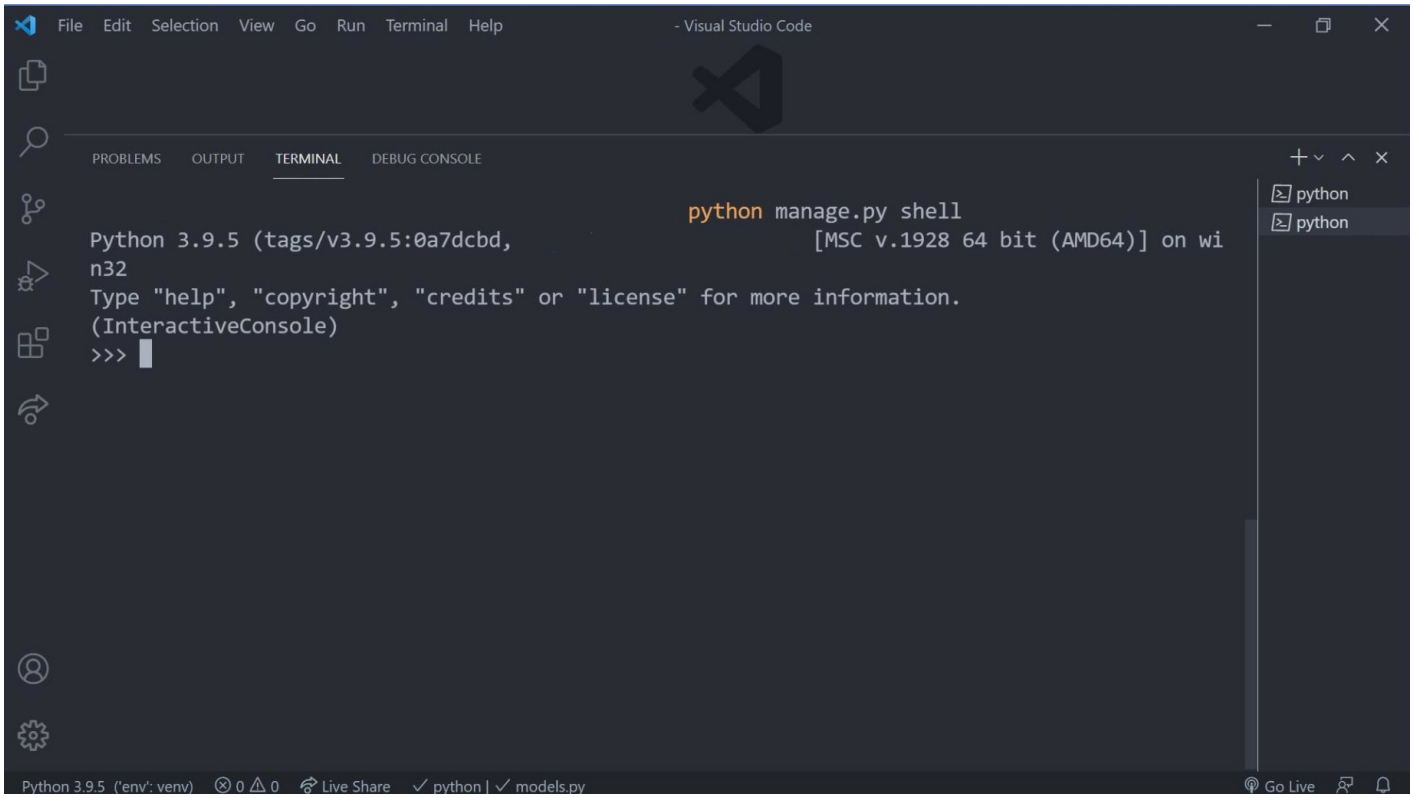
```
#Be sure to import a model first before making queries
from .models import ModelName

#all() - Retrieves all objects from table
queryset = ModelName.objects.all()

#get(attribute='value') - Retrieve a single object based on matched attribute
queryItem = ModelName.objects.get(attribute='value')

#filter(attribute='value') - Returns all items from table that match a particular attribute value
queryset = ModelName.object.filter(attribute='value')
                           .filter(attribute__startswith='value')
                           .filter(attribute__contains='value')
                           .filter(attribute__icontains='value')
                           .filter(attribute__gt='value')
                           .filter(attribute__gte='value')
                           .filter(attribute__lt='value')
                           .filter(attribute__lte='value')

#exclude(attribute='value') - Excludes any object matching a particular filter
queryset = ModelName.object.exclude(attribute='value')

#order_by() - Order a queryset by a particular attribute. Multiple parameters are allowed
queryset = Project.object.filter(title=”first project”).order_by('value1', 'value2')

#order can be reversed by adding "-" before the attribute name
queryset = Project.object.filter(title=”first project”).order_by('-value', '-value2')

#create() - Create an instance of a model
item = ModelName.objects.create(attribute='value')

#save() - Save changes made to a particular object
item = ModelName.objects.get(attribute='value')
item.title = "New Value"
item.save()

#delete() - Deletes a particular object
item = ModelName.objects.last()
item.delete

#Query Models Children
item = ModelName.object.first()
item.childmodel_set.all()

#Query ManyToMany Fields

item = ModelName.object.first()
item.relationshipname.all()

#Add ManyToMany Field
item = ModelName.object.first()
otheritem = OtherModule.objects.create(attribute='value')
item.relationshipname.add(otheritem)
```

Interactive Shell with the Database

In your IDE, open up the terminal window and type:

```
python manage.py shell
```

and you should gain access to the python interpreter or interactive console



We'll run some queries in order to get used to the idea that is not necessary to access the data throught the admin-panel in order to work with our database.

Accessing all records in the Project table with all()

Type in the interactive interpreter command line:

```
from project.models import Project
projects = Project.objects.all()
print(projects)
```

You should get a response with a Queryset object: <Queryset [<Project: ...>]> a list of objects type Project. Let's access the list:

```
for project in projects:
    print(project.title)
    print(project.created)     # This has to be same spelling as typed in the
model for the creation date
```

You should be able to access the attributes of each record in the database brought by the Queryset object.

Accessing individual records with get()

Type in the interactive interpreter command line:

```
projectobj = Project.objects.get(title="Ecommerce Website")
print(projectobj.title)
print(projectobj.created)     # This has to be same spelling as typed in the
model for the creation date
```

Again, you should be able to get data from the Queryset object, but from a single record in this case.

Getting multiple records with filter()

Let's type:

```
projects = Project.objects.filter(vote_ratio__gte=50)
print(f'Project with ratio greater than 50\%:')
for project in projects:
    print(f"{project:< 20}")
```

Now you should get a list of projects with a vote ratio > 50%. Try to write the same query but for any project with vote ration < 50

Accesing Dependant Data in a ManytoOne relationship

Go to your admin-panel and for the Ecommerce Website add a couple of 'up' votes and a single 'down' vote.

Now, go back to your interactive shell and type:

```
project = Project.objects.get(title="Ecommerce Website")
print(project.review_all.set())
```

You should be able to see all the votes in the review table for the "Ecommerce website" similar to this

```
<QuerySet [<Review: up>, <Review: up>, <Review: down>, <Review: up>]>
```

Accesing Dependant Data in a ManytoMany relationship

In this case we'll use the Tag object linked to the Project object. Since the relationship is already set up from previous guides, the only thing needed is to type:

```
tags = project.tag.all()    # It only requires the relationship name (tag for
Tag table)
print(tags)
```

should return something like

```
<QuerySet [<Tag: Django>, <Tag: Python>]>
```