

Tutorial 3

Django Shell

In earlier lab exercises we have used Django admin to interact with the database and then we created a CRUD application to interact with the database through our web application. We can also use the Python shell to create/edit and delete objects in the database. In this exercise we will create a project with a database model and then look at the commands in Python shell for database operations. We can also check the result of our operations using **DBBrowser**.

You can install DB Browser on your home computer/laptop – the download is available at the following link: <https://sqlitebrowser.org/dl/>

Initial Set Up

1. Open **Windows Command Line**, move into the directory called **django projects** and activate the virtual environment
2. Create a new folder called **tutweek4**
3. Move into the **tutweek4** directory and create a virtual environment
4. Create a new Django project called **mysite**
5. Verify that the Django project works by running the server
6. If you visit <http://127.0.0.1:8080/> you should see the familiar Django welcome page

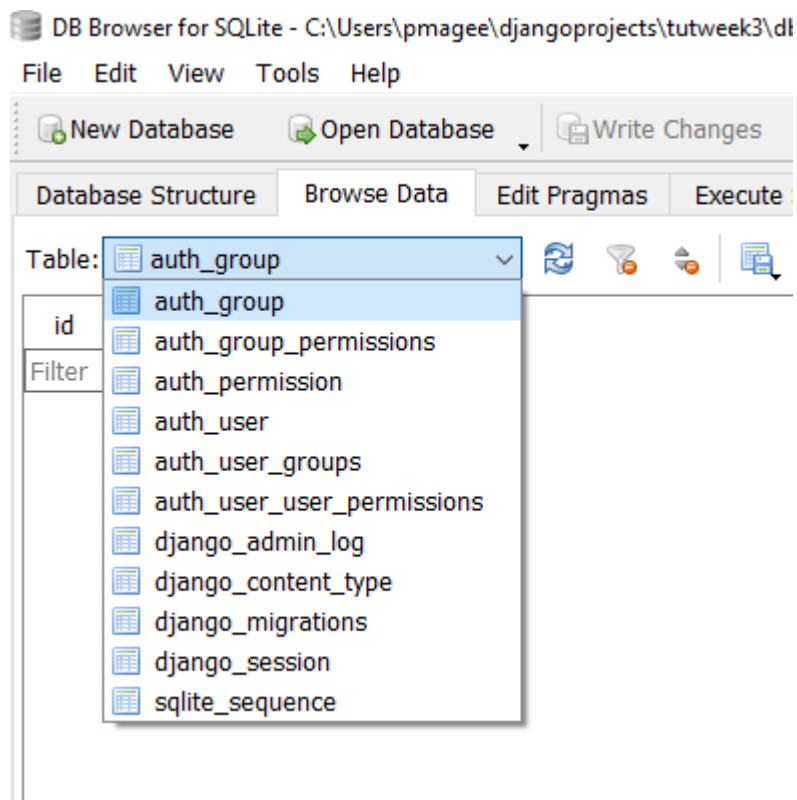
The output in the command line shows a warning about “18 unapplied migrations” although this warning has no effect on the project at this point. Django is letting us know that we have not yet “migrated” or configured our initial database.

You can remove the warning by running the migrate command as shown here:

python manage.py migrate

When you run this command, you will see in the output that 18 migrations are applied. Launch **DB Browser** and open the db.sqlite3 file located in the **tutweek4** folder.

Select the **Browse Data** tab and you will see a list of database tables that already exist in your project. The first 6 tables are part of the auth app which deal with the authentication of users in the Django application.



When you create a database model and run the makemigrations and migrate commands you can view your newly created table in **DB Browser**. Close the database and go back to the command line.

A New App

1. Create an app called **poll**.
2. Open the project in VS Code and add the new app to **INSTALLED_APPS** in the **settings.py** file

Database Models

In the **poll** app, create two models: **Question** and **Choice** using the code provided. A Question has a question and a publication date. A Choice has two fields: the text of the choice and a vote tally. Each Choice is associated with a Question through a foreign key.

1. In VS Code open the **mysite** folder and within the **poll** folder open the file **models.py** and enter the code below:

Code

```
poll > models.py
1  from django.db import models
2
3
4  class Question(models.Model):
5      question_text = models.CharField(max_length=200)
6      pub_date = models.DateTimeField('date published')
7
8
9  class Choice(models.Model):
10     question = models.ForeignKey(Question, on_delete=models.CASCADE)
11     choice_text = models.CharField(max_length=200)
12     votes = models.IntegerField(default=0)
```

Now that our new database model is created, we need to create a new migration record for it and migrate the change into our database.

2. Stop the server and run the makemigrations command

Command Line

```
python manage.py makemigrations poll
```

You should see something like the following:

```
Migrations for 'poll':
poll\migrations\0001_initial.py
- Create model Question
- Create model Choice
```

By running makemigrations, you're telling Django that you've made some changes to your models (in this case, you've made new ones) and that you'd like the changes to be stored as a migration.

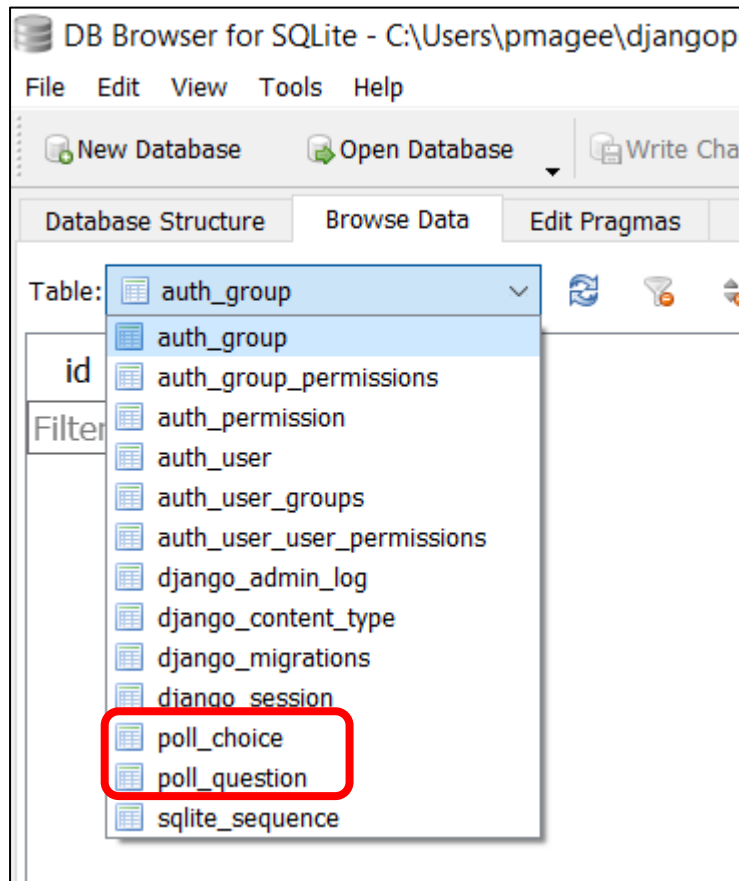
Migrations are how Django stores changes to your models (and thus your database schema).

Now, run migrate again to create the model tables in your database:

Command Line

python manage.py migrate

Open the database file in **DB Browser** again and look for these 2 new tables.



Using the API

Now, let's go into the interactive Python shell and see what the free API Django gives us. To invoke the Python shell, use this command:

python manage.py shell

You should see the three **greater than** signs which means that you are now in the interactive.

```
Python 3.10.7 (tags/v3.10.7:6cc6b13, Sep  5 2022, 14:08:36) [MSC v.1933 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>>
```

To query the database models from here we need to import them:

```
>>>from poll.models import Choice, Question
```

To check for questions in the database, try the following command:

```
>>>Question.objects.all()
```

```
>>> Question.objects.all()  
<QuerySet []>
```

The result of this query is a queryset which represents a collection of objects from your database. In this case the queryset is empty because we have not added any questions yet.

We will add a question now which has two fields: the question itself and the date of publication. For the date of publication, we will use the current date and time but to set this value we need to import the Django utility **timezone**.

```
>>>from django.utils import timezone
```

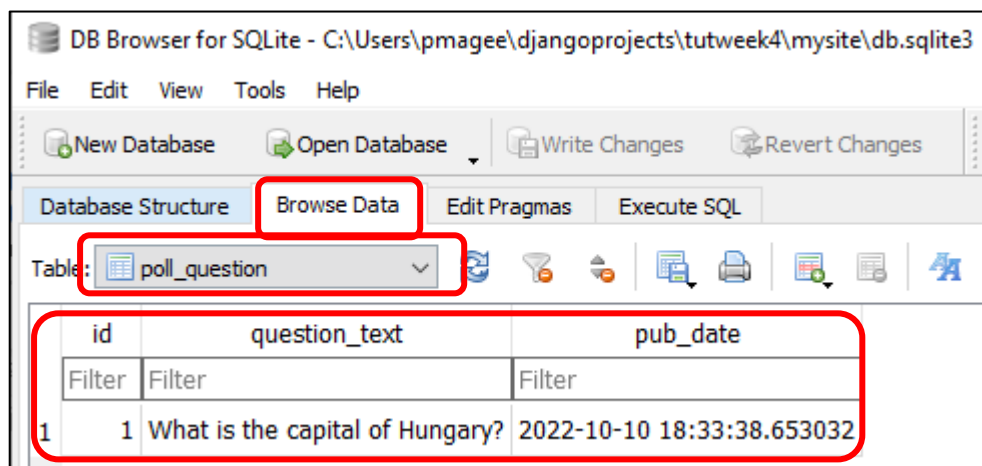
Now we can add a question

```
>>>q = Question(question_text="What is the capital of Hungary?",  
pub_date=timezone.now())
```

Save the object into the database. You must call `save()` explicitly.

```
>>>q.save()
```

Check the Question table in DB Browser for this new object



The screenshot shows the DB Browser for SQLite application. The title bar indicates the database file is 'C:\Users\pmagee\django\projects\tutweek4\mysite\db.sqlite3'. The 'Browse Data' tab is selected, and the 'poll_question' table is chosen from the 'Table:' dropdown. A red rectangle highlights the table name in the dropdown. Another red rectangle highlights the data table, which has three columns: 'id', 'question_text', and 'pub_date'. Each column has a 'Filter' input field above it. The first row of data shows 'id' as 1, 'question_text' as 'What is the capital of Hungary?', and 'pub_date' as '2022-10-10 18:33:38.653032'.

id	question_text	pub_date
1	What is the capital of Hungary?	2022-10-10 18:33:38.653032

We can also check the id of the question in the shell & we will see that the id is 1

```
>>>q.id
```

Output

```
>>> q.id
1
>>>
```

We can also access model field values via Python attributes.

```
>>>q.question_text
```

Output

```
>>> q.question_text
'What is the capital of Hungary?'
>>>
```

```
>>>q.pub_date
```

Output

```
>>> q.pub_date
datetime.datetime(2022, 10, 10, 18, 33, 38, 653032, tzinfo=datetime.timezone.utc)
>>>
```

We can change values by changing the attributes, then calling save().

```
>>>q.question_text = "What is the capital of ireland?"
```

```
>>>q.save()
```

```
>>>q.question_text
```

Output

```
>>> q.question_text = "What is the capital of Ireland?"
>>> q.save()
>>> q.question_text
'What is the capital of Ireland?'
```

To display all the questions in the database we use objects.all()

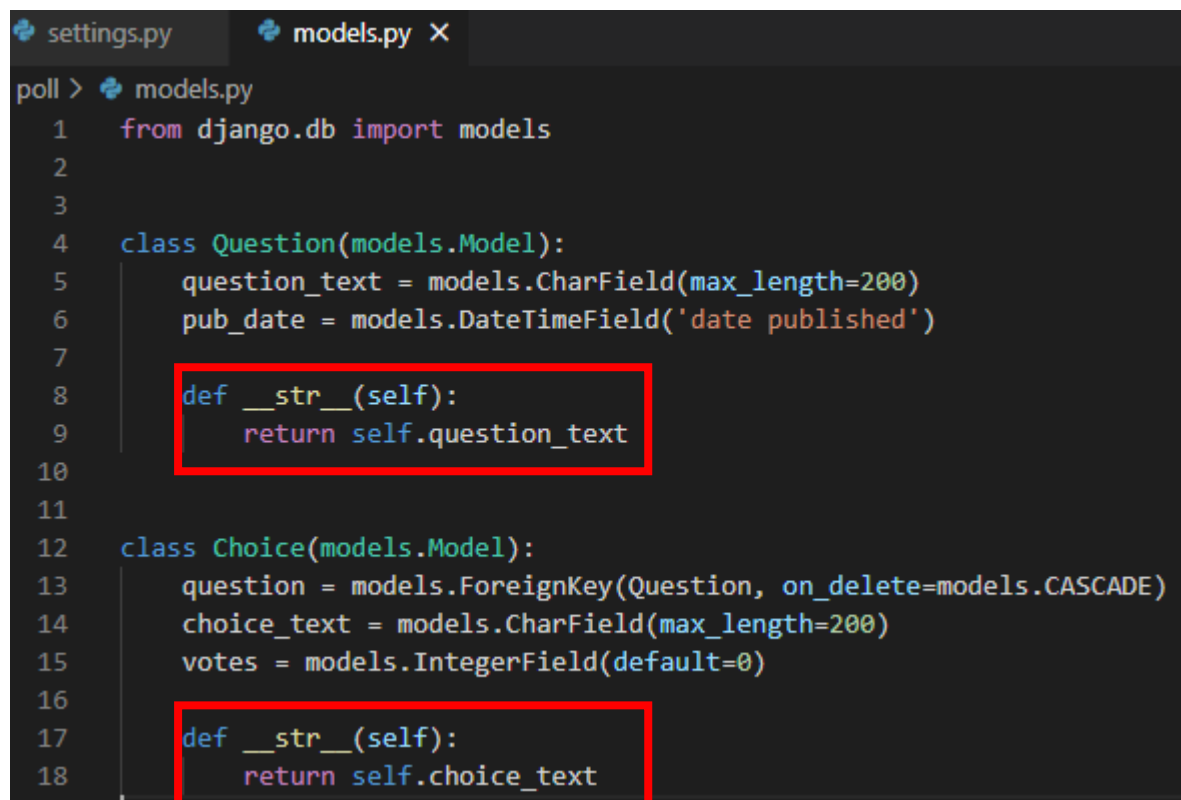
```
>>>Question.objects.all()
```

Output

```
>>> Question.objects.all()
<QuerySet [<Question: Question object (1)>]>
```

The output <Question: Question object (1)> isn't very user friendly but we can fix that by editing the Question model (in the poll/models.py file) and adding a __str__() method to both Question and Choice.

Add the following code to **models.py**:



```
poll > models.py
1  from django.db import models
2
3
4  class Question(models.Model):
5      question_text = models.CharField(max_length=200)
6      pub_date = models.DateTimeField('date published')
7
8      def __str__(self):
9          return self.question_text
10
11
12  class Choice(models.Model):
13      question = models.ForeignKey(Question, on_delete=models.CASCADE)
14      choice_text = models.CharField(max_length=200)
15      votes = models.IntegerField(default=0)
16
17      def __str__(self):
18          return self.choice_text
```

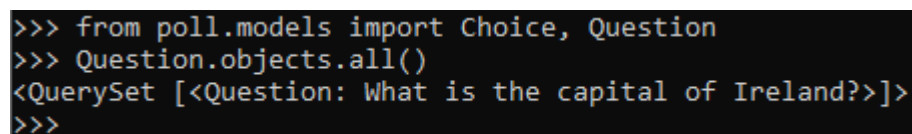
Save these changes and in order to see the effect of these changes exit out of the current shell by typing **exit()** and then open a new shell by running **python manage.py shell** again.

Try these commands again and note the output looks a bit better:

```
>>> from poll.models import Choice, Question
```

```
>>> Question.objects.all()
```

Output

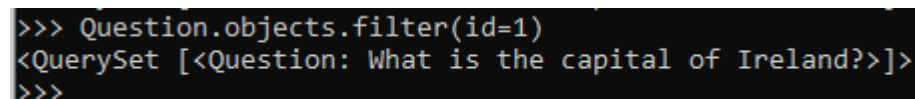


```
>>> from poll.models import Choice, Question
>>> Question.objects.all()
<QuerySet [<Question: What is the capital of Ireland?>]>
>>>
```

Django provides a rich database lookup API that's entirely driven by keyword arguments. The following query filters the objects based on the id value supplied:

```
>>> Question.objects.filter(id=1)
```

Output



```
>>> Question.objects.filter(id=1)
<QuerySet [<Question: What is the capital of Ireland?>]>
>>>
```

The following query filters the objects based on the question that starts with "What"


```
>>> Question.objects.filter(question_text__startswith='What')
```

Output

```
>>> Question.objects.filter(question_text__startswith='What')
<QuerySet [<Question: What is the capital of Ireland?>]>
>>>
```

To retrieve the questions that were published this year, we import `timezone`, find the current year and then use the `get` function to find a match with the current year and the year attribute of the `pub_date` field.

```
>>> from django.utils import timezone
>>> current_year = timezone.now().year
>>> Question.objects.get(pub_date__year=current_year)
```



This is a double under score

Output

```
>>> Question.objects.get(pub_date__year=current_year)
<Question: What is the capital of Ireland?>
```

If we request an ID that doesn't exist, this will raise an exception.

```
>>> Question.objects.get(id=2)
```

Output

```
>>> Question.objects.get(id=2)
Traceback (most recent call last):
  File "<console>", line 1, in <module>
  File "C:\Users\pmagee\django\projects\tutweek4\env\lib\site-packages\django\db\models\manager.py", line 85, in manager_method
    return getattr(self.get_queryset(), name)(*args, **kwargs)
  File "C:\Users\pmagee\django\projects\tutweek4\env\lib\site-packages\django\db\models\query.py", line 650, in get
    raise self.model.DoesNotExist(
poll.models.Question.DoesNotExist: Question matching query does not exist.
```

Lookup by a primary key is the most common case, so Django provides a shortcut for primary-key exact lookups. The following is identical to `Question.objects.get(id=1)`.

```
>>> Question.objects.get(pk=1)
```

Output

```
>>> Question.objects.get(pk=1)
<Question: What is the capital of Ireland?>
```

Now we will give the `Question` a couple of `Choices`. The `create` call will construct a new `Choice` object, does the `INSERT` statement, adds the choice to the set of available choices and returns the new `Choice` object. Django creates a set to hold the "other side" of a `ForeignKey` relation (e.g. a question's choice) which can be accessed via the API.

Retrieve the Question object first:

```
>>> q = Question.objects.get(pk=1)
```

Display any choices from the related object set - none so far.

```
>>> q.choice_set.all()
```

Output

```
>>> q.choice_set.all()
<QuerySet []>
```

Create three choices.

```
>>> q.choice_set.create(choice_text='Dublin', votes=0)
```

```
>>> q.choice_set.create(choice_text='Cork', votes=0)
```

For the third choice we assign it to an object called c

```
>>> c = q.choice_set.create(choice_text='Galway', votes=0)
```

Choice objects have API access to their related Question objects.

```
>>> c.question
```

Output

```
>>> c.question
<Question: What is the capital of Ireland?>
```

And vice versa: Question objects get access to Choice objects.

```
>>> q.choice_set.all()
```

Output

```
>>> q.choice_set.all()
<QuerySet [<Choice: Dublin>, <Choice: Cork>, <Choice: Galway>]>
```

We can find the count of objects in our set.

```
>>> q.choice_set.count()
```

Output

```
>>> q.choice_set.count()
3
```

The API automatically follows relationships as far as you need. Use double underscores to separate relationships. Find all Choices for any question whose pub_date is in this year reusing the 'current_year' variable we created above.

This is a double under score

```
>>> Choice.objects.filter(question__pub_date__year=current_year)
```

Output

```
>>> Choice.objects.filter(question__pub_date__year=current_year)
<QuerySet [<Choice: Dublin>, <Choice: Cork>, <Choice: Galway>]>
```

Let's delete one of the choices. Use the delete() function for that.

```
>>> c = q.choice_set.filter(choice_text__startswith='Gal')
```

```
>>> c.delete()
```

This is a double under score

```
>>> c = q.choice_set.filter(choice_text__startswith='Gal')
>>> c.delete()
(1, {'poll.Choice': 1})
```

The delete() function returns the number of objects deleted and a dictionary with the number of deletions per object type. In this case the 1 inside the round brackets denotes 1 Choice object deleted and the dictionary {'poll.Choice':1} denotes 1 Choice deletion for the Poll object.

View the set of choices again and we see that there are only 2 left:

```
>>> q.choice_set.all()
```

```
>>> q.choice_set.all()
<QuerySet [<Choice: Dublin>, <Choice: Cork>]>
```

Check the database in DB Browser again to see the entries in the Choice table.

Create a private repo in your GitHub account and push this code to it using the appropriate set of commands.