Theory: Object modification with ORM

22 minutes 0 / 5 problems solved

The world changes over time, and so does your data. When you have some new object to store, you ***create*** it in a database for further use; if you don't need it anymore, you ***delete*** it. When the object's properties change, you make an ***update***. These three actions along with reading are known as **CRUD** (Create, Read, Update, Delete) operations.

Even though Django methods use almost identical naming, let's look at them closer and learn how to use them right.

§1. Create

Learning is hard, so we decide to take a break and make a truly entertaining computer game for ourselves. The galaxy is in danger and our brave space team must prevent the alien invasion. The main weapons of our team are diplomacy and science: you'll see what it means further in the topic. At this time we need only two models to start the game:

```
from django.db import models


class Alien(models.Model):
    type = models.CharField(max_length=32)
    distance_to_galaxy = models.IntegerField()
    threat = models.IntegerField()
    speed = models.IntegerField()




class Weapon(models.Model):

    type = models.CharField(max_length=32)

    quantity = models.IntegerField()

    power = models.IntegerField()

    coverage_distance = models.IntegerField()
```

First, we should create some weapons for our heroes. Let's equip them with ***eloquence*** to convince aliens to go away and ***jammers*** to modify scouting signals:

```
eloquence = Weapon.objects.create(
   type='eloquence', power=100, coverage_distance=100, quantity=10
)

jammers = Weapon(type='jammer', power=10, coverage_distance=1000, quantity=50)
jammers.save()
```

Two methods are shown to illustrate how you can create new objects in a database. We create ***eloquence*** with the help of Object Manager. It has a method `create`, to which you can pass all the parameters your object has. If the result of this operation is successful, the call saves the object to the database and returns the instance of a `Weapon` class.

The second method is to create an instance of a class and then save it manually. The two methods are pretty much the same, so you can use whichever you like most.

> When you create an instance manually, the object will not be saved to a database until you call the save method explicitly.

Current topic:

Object modification with ORM

Topic depends on:

✕ Queries and filters

Table of contents:

§2. Delete

The alien invasion is coming. Meanwhile, we're really hoping that our game will attract and satisfy a lot of users. For each session, we'll create hundreds or even thousands of aliens to make the battle hard enough. If we don't clear the database of all the defeated aliens, operations will become slower and we will eventually run out of disk space.

The first enemy comes from a nearby galaxy located only 23 solar years from ours:

```
et_alien = Alien.objects.create(type='ET', distance_to_galaxy=23, threat=70, speed=5)
```

In five moves it passes the border, so the player can apply eloquence to deceive the opponent and make it turn in the opposite direction. The power of eloquence is 100 and the threat is 70, so in one move the player can resist the first invasion!

We do not need the `et_alien` anymore, so let's delete it:

```
et_alien.delete()
```

That's simple. We call the `delete` method of an instance of `Alien` class and the object is deleted from the database.

> The delete method removes the object from a database but does not delete an instance of a class. Do not use the object after this operation to prevent ambiguity.

The game continues...

§3. Update

We have two main powers of modifying objects: to *create* and *delete* them. The third power is to change an existing object. When the properties of an object change, we should update them in the database.

The next enemy of the galaxy is a Predator, an opponent that can hardly be defeated in a single move since he comes from deep space and our weapons are not strong enough.

```
predator = Alien.objects.create(type='Predator', distance_to_galaxy=550, threat=40, speed=30)
```

Our jammers make a barrier for signals through space, so the enemy loses direction. The player applies this weapon on the next move. The number of jammers decreased by one and simultaneously the threat diminished:

```
jammers.quantity -= 1
jammers.save()

predator.distance_to_galaxy -= predator.speed
predator.threat -= jammers.power
predator.save()
```

Updating an object is a two-step operation. We change the attributes of an object and then we call the `save` method as for manual creation of an object.

> The attributes of an instance are saved only in a Python object until you call the save method. Your object is not an exact replica of a database row in a current moment, so remember to save it to synchronize the changes you made.

In three more moves, the player defeats the Predator with jammers and it flies away in an unknown direction. We call `predator.delete()` and go forward to the next round.

§4. Modification of QuerySet

We can modify each object as we like, but remember: aliens may come in a swarm. Can we apply our weapons to all of them simultaneously? With the approach, we just considered, the answer would be *no.* If only we could use a QuerySet for this task... The good news is, we surely can!

Space bugs come in a pack of three:

```
for _ in range(3):

 Alien.objects.create(type='Space Bug', distance_to_galaxy=30, threat=150, speed=12)
```

At the first move the player applies *eloquence*:

```
eloquence.quantity -= 1
eloquence.save()

space_bug = Alien.objects.filter(type='Space Bug').first()

Alien.objects.filter(type='Space Bug').update(
   distance_to_galaxy=space_bug.distance_to_galaxy - space_bug.speed,
   threat=space_bug.threat - eloquence.power
)
```

The bugs are not defeated yet, but we update their position and threat in one call, not three. We get an Object Manager of a model, filter out objects that we want and call `update` on the QuerySet. The syntax is the same: we pass parameters and their new values to the method and Django does the rest of the work with the database.

Finally, the player applies *eloquence* again and the bugs are convinced to stop their invasion. Again, we call the method on a QuerySet:

```
Alien.objects.filter(type='Space Bug').delete()
```

All space bugs are now removed from the database.

> Although the delete method works for both an object and a QuerySet, the update method is defined only for a QuerySet. If you call an update method on an object, you'll get an AttributeError exception.

We have only three rounds, so the game is over. This time, the player wins and saves our galaxy from all space creatures out there. No aliens were harmed during the making of this topic.