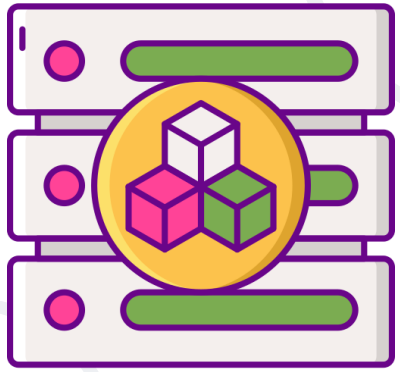
Models in Django - Part 2



SoftUni Team

Technical Trainers

Software University

SoftUni

Software University

https://softuni.bg

Table of Contents

Have a Question?

Software University

sli.do

#python-web

Working with Model Objects

Reading Data

Reading Data from the Database

- **Use Python code** to retrieve data from the database
  - Use the **Manager** on the model class
  - Construct a **QuerySet**

views.py

```
def show_all_departments(request)
    all_departments = Department.objects.all()
    ...
    context = {"departments": all_departments}
    return render(request, 'departments.html', context)
```

Retrieve all objects

Model Manager

- Each model has **at least one Manager**, and it's called **`objects`** by default
- Managers are **accessible** only via **model classes** rather than from model instances

views.py

```
def show_all_departments(request)
    all_departments = Department.objects
    print(all_departments)
    # departments.EmployeeNew.objects
    print(type(all_departments))
    # <class 'django.db.models.manager.Manager'>
```

QuerySet

- QuerySet is a **collection of objects** from the database
- You can work with it without hitting the database
- It hits the database when you **evaluate** the **queryset**

views.py

```
def show_all_departments(request)
    all_departments = Department.objects.all()
    print(all_departments)
    # <QuerySet []>
    print(type(all_departments))
    # <class 'django.db.models.query.QuerySet'>
```

Working with Model Objects

Filtering Data

Filtering Objects

- Return objects that **match** given parameters

```
employees_aged_35 = Employee.objects.filter(age=35)
# returns a QuerySet with all employees (objects) whose age is 35
```

- Return objects that do **NOT match** given parameters

```
employees_not_aged_35 = Employee.objects.exclude(age=35)
# returns a QuerySet with all employees (objects) whose age is NOT 35
```

- Return **only one object** that matches your query

```
employee_with_id_one = Employee.objects.get(id=1)
# returns the employee object with an id equal to 1
```

Filtering with Shortcut Functions

- Get the desired **object** or raise an **HTTP 404**

```
employee_with_id_one = get_object_or_404(Employee, pk=1)
# try to get an employee with an id of 1;
# if not - raise an Http404 exception;
```

- Get the desired **queryset** (casted to list) or raise an **HTTP 404**

```
employees_aged_35 = get_list_or_404(Employee, age=35)
# try to filter a list of employees who are aged 35;
# if the list is empty - raise an Http404 exception;
```

Adding Filter Conditions

- The filtering methods accept key-value paired **field lookup parameters**
- They take the form **`field__lookuptype=value`**

```
Employee.objects.filter(age__lte=35)
# returns a QuerySet with all employees (objects)
whose age is less than or equal to 35
```

```
SELECT * FROM employees WHERE age <= 35;
```

Field Lookup Types (1)

- To match objects with **exactly the given value**

```
Employee.objects.filter(job_level="Junior")
Employee.objects.exclude(job_level__exact="Junior") # explicit form
Employee.objects.get(job_level__iexact="Junior")    # case-insensitive match
```

- To match objects that **contain the given value**

```
Employee.objects.exclude(job_title__contains="Engineer")
Employee.objects.filter(job_title__icontains="engineer") # case-insensitive
```

- To match objects that **starts-with** or **ends-with** the given value

```
Employee.objects.exclude(job_title__startswith="Senior")
Employee.objects.filter(job_title__endswith="Engineer")
```

Field Lookup Types (2)

- To match objects with a value **greater than** the given value

```
Employee.objects.filter(age__gt=20)   # greater than
Employee.objects.exclude(age_gte=20)  # greater than or equal to
```

- To match objects with a value **less than** the given value

```
Employee.objects.filter(age_lt=20)   # less than
Employee.objects.exclude(age_lte=20) # less than or equal to
```

- To match objects **in a given range** (inclusive)

```
Employee.objects.filter(age__range=(20, 30)) # from 20 to 30 both inclusive
```

Chaining Filter Conditions

- **Date/time** field lookup types allow **chaining** additional field lookups

```
Employee.objects.filter(birth_date__year__gt=1999)
# returns a QuerySet with all employees (objects)
who are born after 1999
```

```
SELECT * FROM employees WHERE birth_date > '2000-01-01';
```

More Chained Conditions: https://docs.djangoproject.com/en/4.1/ref/models/querysets/#date

Working with Model Objects

Deleting Data

Deleting Model Object (1)

- The **`delete()`** method **immediately deletes** the model object
- You should **explicitly request** the object
- The method returns:
  - The **number of objects deleted**
  - The **number of deletions per object type**

```
employee = Employee.objects.first()
employee.delete()
# (1, {'departments.Employee': 1})
```

Deleting Model Object (2)

- Delete a **single** object

```
employee = Employee.objects.get(pk=1)
employee.delete()
```

- Delete **multiple** objects in a QuerySet

```
employees = Employee.objects.all()
employees.delete()
```

- Note: when deleting an object with **foreign keys**, it will emulate the behavior of the **SQL constraint ON DELETE**

Class Meta

Class Meta

- To **insert model metadata** in the model, use the inner-class Meta
  - Adding the class is completely **optional**

```
class Employee(models.Model):
    ...
    year_of_employment = models.IntegerField()

    class Meta:
        ordering = ["-year_of_employment"]
```

Meta option

Meta Options (1)

- You can create an **abstract base class** to put some common information into other models
- The model will **not create** any **database** table

```
class CommonInfo(models.Model):
    name = models.CharField(max_length=100)
    ...
    class Meta:
        abstract = True

class Employee(CommonInfo):
    ...
```

Meta Options (2)

- You can generate a **default ordering** for the object when retrieving data from the model

```
class Employee(models.Model):
    ...

    class Meta:
        ordering = ['name', '-years_of_employment']
```

Ascending order

Descending order

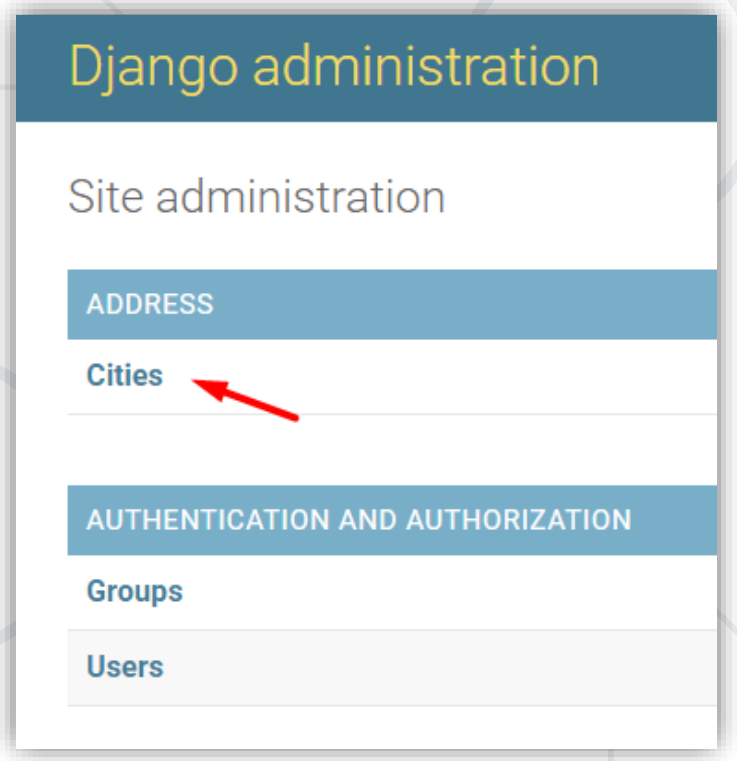
- Note: if you order by a field that is **not unique**, objects with the **same values** may appear in a **different order**

Meta Options (3)

- You can give a **plural name** to your model to be represented correctly in the Django admin site

```
class City(models.Model):
    ...

    class Meta:
        verbose_name_plural = "cities"
```



More Meta Options: https://docs.djangoproject.com/en/4.1/ref/models/options/

Model Methods

Custom Model Methods

- Add functionality on a **particular** model instance
- Keeps **business logic** in one place

```
class Employee(models.Model):
    ...

    def years_of_experience:
        # returns the years and months of experience
```

Custom Model Properties

- Add **managed attributes**
- Feature of Python

```
class Employee(models.Model):
    ...

    @property
    def full_name:
        # returns the full name of an employee
```

Built-in Model Methods

- Each of them have special purpose

```
class Employee(models.Model):
    ...

    def get_absolute_url(self):
        # returns a calculated URL for that object

    def __str__(self):
        # returns a human-readable representation
```

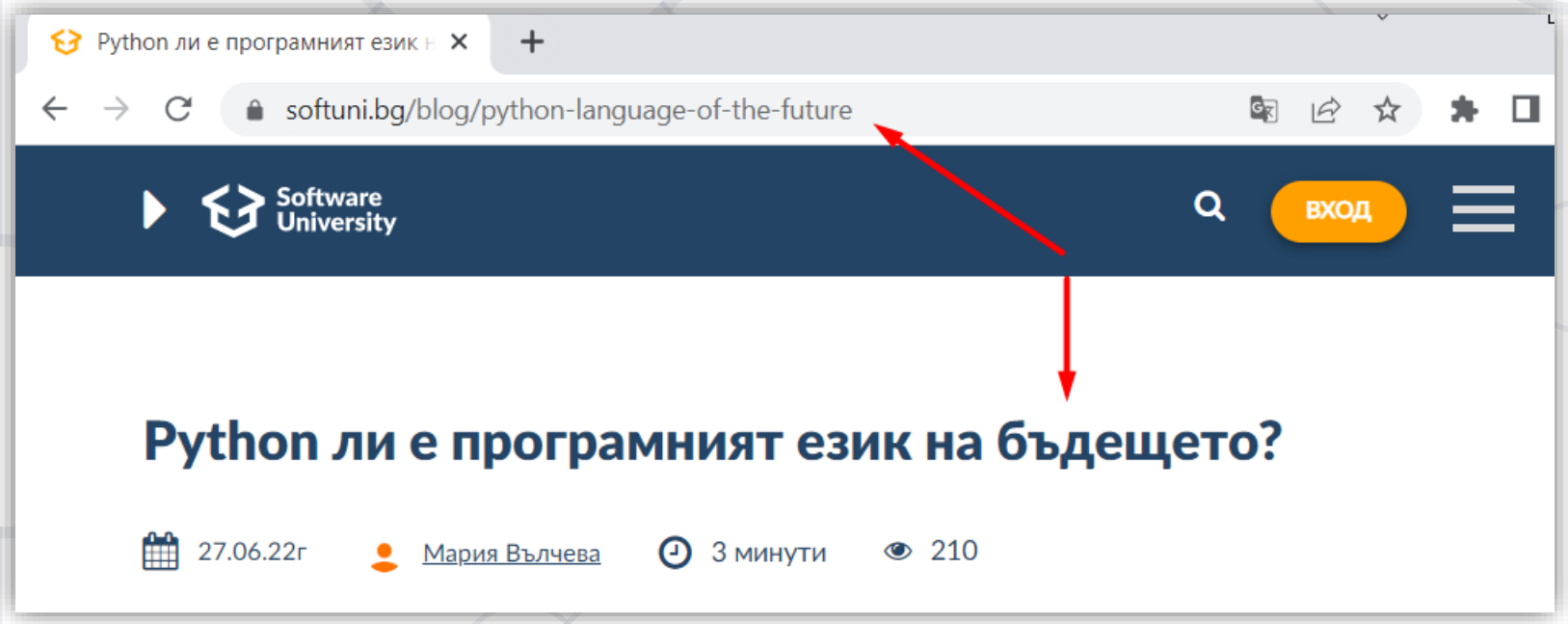
More Model Methods: https://docs.djangoproject.com/en/4.1/topics/db/models/#model-methods

Adding Slugs

Live Demo

Using Slugs

- Slugs are generally **used in URLs**
- It is often used to **automatically prepopulate** a slug with a value based on another value



Add a SlugField

- Define a **field** where the data of each slug will be stored
- Each value should be **unique** as it is part of an URL

departments/models.py

```
class Department(models.Model):
    name = models.CharField(max_length=100)
    ...

    slug = models.SlugField(unique=True)
```

Add the URL pattern

- The route should be a **slug value** which will be passed to the view function

departments/urls.py

```
urlpatterns = [
    ...
    path("<slug:slug>", views.show_department, name="show-dep"),
]
```

Create the View

- Create the view as usual
- Try to find to object by the given slug

departments/views.py

```
def show_department(request, slug):
    department = get_object_or_404(Department, slug=slug)
    context = {"department": department}
    return render(request, "department_details.html", context)
```

Generate the Path Reference

- Define the built-in model **method** that calculates the absolute URL
- Use the **reverse()** function
- Pass the slug value as an **argument**

departments/models.py

```
class Department(models.Model):
    ...

    def get_absolute_url(self):
        return reverse("show-dep", kwargs={"slug": self.slug})
```

Add the Slug in the Template

- Specify the **URL** for each model
  - Use the **generated path** in the model

dashboard.html

```
<h1>All Departments</h1>
{% for dep in departments_list %}
  <ul>
    <li><a href="{{ dep.get_absolute_url }}">{{ dep.name }}</a></li>
  </ul>
{% endfor %}
```

Customize the Admin Site

- The slug field should be **prepopulated** with the value in the name field

departmets/admin.py

```
class DepartmentAdmin(admin.ModelAdmin):
    prepopulated_fields = {"slug": ("name",)}

admin.site.register(Department, DepartmentAdmin)
```

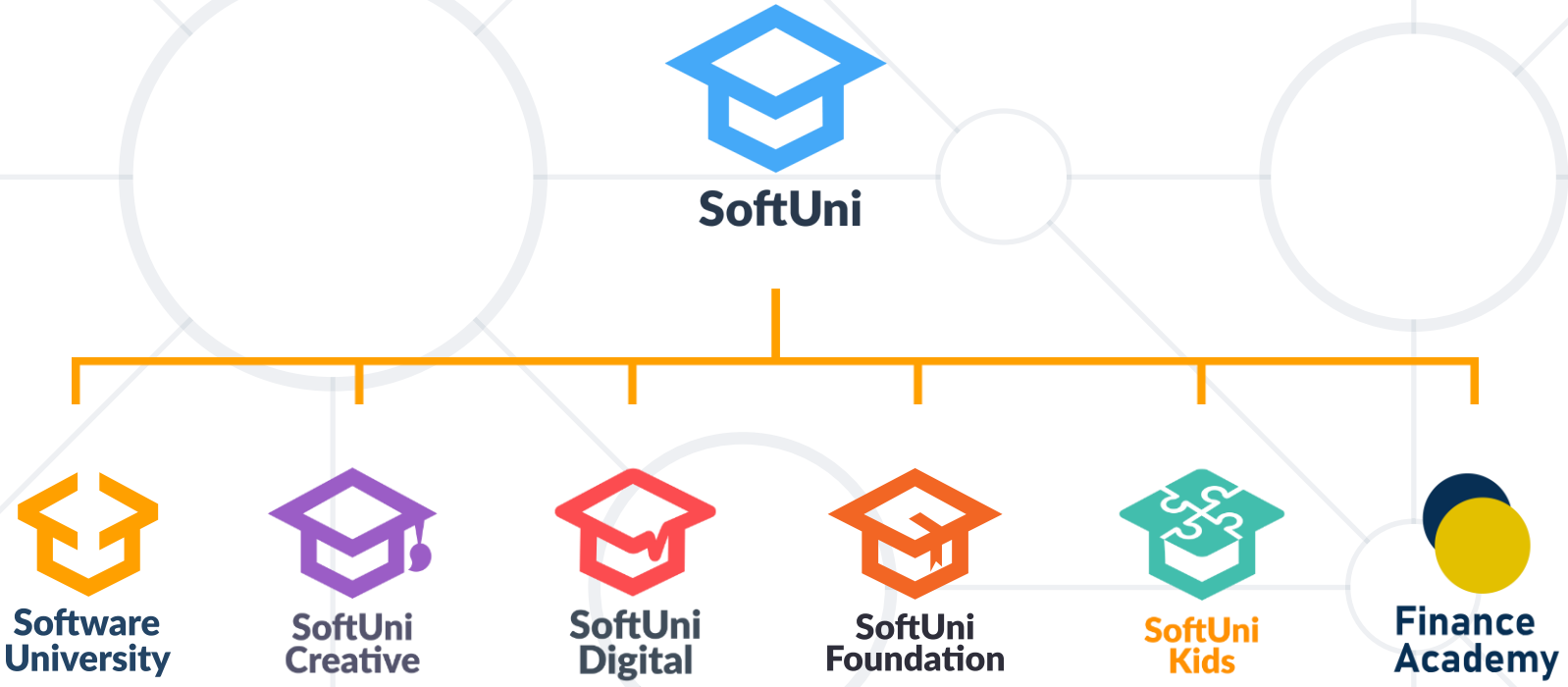
Live Demo

Live Exercises in Class

Summary

- **Models** allow us to work with data using Python code
- Use **class Meta** to insert "anything that's not a field"
- Use model **methods** to add custom "row-level" functionality to the objects

Questions?



SoftUni Diamond Partners

Software University

SUPER HOSTING .BG

Coca-Cola HBC Bulgaria

POKERSTARS
POKER | CASINO | SPORTS
a Flutter International brand

INDEAVR
Serving the high achievers

AMBITIONED

DRAFT KINGS

SOFTWARE GROUP

createX

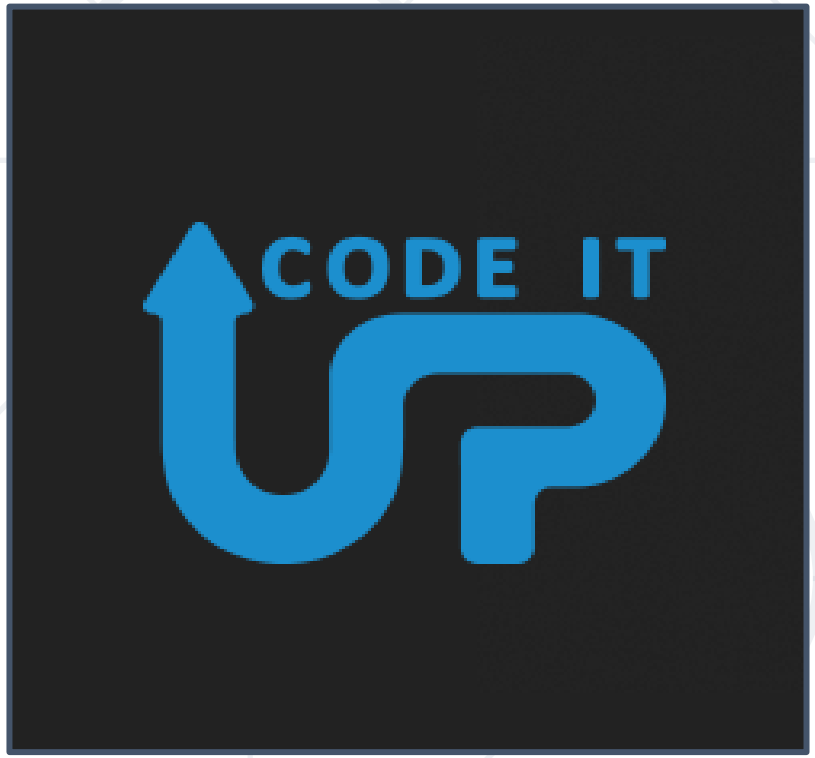
Postbank
Решения за твоето утре

BOSCH

DXC TECHNOLOGY

SmartIT

Educational Partners



License

- This course (slides, examples, demos, exercises, homework, documents, videos, and other assets) is **copyrighted content**
- Unauthorized copy, reproduction, or use is illegal
- © SoftUni – https://softuni.org
- © Software University – https://softuni.bg

Trainings @ Software University (SoftUni)

- Software University – High-Quality Education, Profession and Job for Software Developers
  - softuni.bg, softuni.org
- Software University Foundation
  - softuni.foundation
- Software University @ Facebook
  - facebook.com/SoftwareUniversity
- Software University Forums
  - forum.softuni.bg