SDEV 2401

Rapid Backend App Development

Databases, Models, Migrations and the Admin - 1

Expectations - What I expect from you

- No Late Assignments

- No Cheating

- Be a good classmate

- Don't waste your time

- Show up to class

Agenda

On the right is what we will cover today.

- Django ORM: Relationships with ForeignKey and ManyToMany
- Example: Author and Book Models
- Making and Applying Migrations
- Adding Data in the Admin
- Displaying Related Data in a Template
- Filtering by Related Fields
- ManyToMany Relationships
- Summary

Django ORM: Relationships with ForeignKey and ManyToMany

Django's ORM lets you define relationships between models, such as one-to-many and many-to-many, using special field types.

- We'll use a `Library` example with `Book` and `Author` models.
- You'll see how to set up ForeignKey and ManyToMany relationships and display related data in templates.

Example: Author and Book Models

On the right, we’ll define an `Author` model and a `Book` model. Each book is written by one author (ForeignKey), and each book can have multiple genres (ManyToMany).

Making and Applying Migrations

After defining your models, run these commands to create and apply migrations:

```
python manage.py makemigrations
python manage.py migrate
```

This will create the necessary tables and relationships in your database.

Adding Data in the Admin

- Register your models in `admin.py` to manage them via the Django admin interface.
- Example:

```python
from django.contrib import admin
from .models import Author, Book, Genre

admin.site.register(Author)
admin.site.register(Book)
admin.site.register(Genre)
```

- Add some authors, genres, and books in the admin.

Displaying Related Data in a Template

You can fetch all books and display their authors and genres in a template.

```python
# views.py
from django.shortcuts import render
from .models import Book

def book_list(request):
    books = Book.objects.all()
    return render(request, 'library/book_list.html', {'books': books})
```

Displaying Related Data in a Template Continued

To display the books in your template, you can loop through the `books` context variable passed from the view. Each book will show its title, author, published year, and associated genres.

In your template:

```
<!-- library/book_list.html -->
<ul>
  {% for book in books %}
    <li>
      <strong>{{ book.title }}</strong> by {{ book.author.name }} ({{ book.published_year }})
      <ul>
        {% for genre in book.genres.all %}
          <li>{{ genre.name }}</li>
        {% endfor %}
      </ul>
    </li>
  {% endfor %}
</ul>
```

Filtering by Related Fields

You can filter books by author or genre using related fields.

```
# Get all books by a specific author
books_by_alice = Book.objects.filter(author__name="Alice Smith")

# Get all books in the "Science Fiction" genre
scifi_books = Book.objects.filter(genres__name="Science Fiction")
```

ManyToMany Relationships

- ManyToMany fields let you associate multiple records from one model with multiple records from another.
- In the example, a book can have many genres, and a genre can belong to many books.
- You can access related objects using `.all()` in templates and views.

Summary

- Use `ForeignKey` for one-to-many relationships (e.g., each book has one author).
- Use `ManyToManyField` for many-to-many relationships (e.g., books and genres).
- Access related data in templates using dot notation and `.all()`.
- Filter and display related data for more dynamic web pages.

NAIT

Example

Let’s go do an example together