DJANGO

Day 77

📌 Models & Database

1. ✅ What is ORM?
2. ✅ Define Models in `models.py`
3. ✅ Migrations: `makemigrations` & `migrate`
4. ✅ SQLite – Django's default database
5. ✅ Registering models in the admin panel *(bonus)*

🧠 1. What is ORM?

ORM (Object-Relational Mapping) allows you to interact with the database using **Python classes and objects**, instead of SQL queries.

🔄 Example:

Without ORM (Raw SQL):

```
SELECT * FROM users WHERE id = 1;
```

With Django ORM:

```
User.objects.get(id=1)
```

👉 Django uses its ORM to map:

- **Models (Python classes)** → **Tables in the database**
- **Model fields (class attributes)** → **Table columns**

2. Define Models in `models.py`

Let's create a simple model called `Product`.

☑ In `myapp/models.py`:

```
from django.db import models

class Product(models.Model):
    name = models.CharField(max_length=100)
    price = models.DecimalField(max_digits=8, decimal_places=2)
    description = models.TextField()
    in_stock = models.BooleanField(default=True)

    def __str__(self):
        return self.name
```

🔍 Explanation:

Field Type	Purpose
`CharField`	Short string (max length needed)
`DecimalField`	For prices, with decimal places
`TextField`	Large text content
`BooleanField`	True/False (default is True)

⚙ 3. Migrations: `makemigrations` & `migrate`

☑ Step 1: Make migrations (generate migration file)

```
python manage.py makemigrations
```

You'll see:

```
Migrations for 'myapp':
  myapp/migrations/0001_initial.py
    - Create model Product
```

☑ Step 2: Apply migration (update database)

```
python manage.py migrate
```

You'll see:

```
Applying myapp.0001_initial... OK
```

🧱 4. SQLite (Default Database)

Django uses **SQLite** by default. It's:

- Lightweight
- Easy to use
- Suitable for development and small projects

You'll find a file created:

`db.sqlite3`

☑ Optional: View Database (using DB browser)

You can open `db.sqlite3` using a tool like:

- **DB Browser for SQLite**
- Or use the Django shell:

```
python manage.py shell
 from myapp.models import Product
Product.objects.all()
```

☑ 5. Bonus: Register Model in Django Admin

To manage your `Product` model via Django admin:

Step 1: Register the model

In `myapp/admin.py`:

```
from django.contrib import admin
from .models import Product

admin.site.register(Product)
```

Step 2: Create superuser (if not already)

```
python manage.py createsuperuser
```

Step 3: Run the server and visit:

http://127.0.0.1:8000/admin/

Log in with your superuser credentials — now you can **add/edit/delete `Product`** entries using the UI.

☑ Summary

Command	Purpose
`python manage.py makemigrations`	Generate migration files (Python → SQL)
`python manage.py migrate`	Apply those migrations to the DB
`python manage.py createsuperuser`	Create admin user for backend access
`Product.objects.all()`	ORM query to fetch all Product records

🔧 Connect Django with MySQL

☑ Step 1: Install MySQL

Install **MySQL Server** and **MySQL Workbench** (optional GUI) from:

🔗 https://dev.mysql.com/downloads/

☑ Step 2: Install MySQL client for Python

In your virtual environment, run:

```
pip install mysqlclient
```

⚠️ On Windows, you may need to install Microsoft Visual C++ Build Tools if there's an error.

Alternative (cross-platform):

```
pip install pymysql
```

And in your `myproject/__init__.py` file, add:

```
import pymysql
pymysql.install_as_MySQLdb()
```

☑ Step 3: Create MySQL Database

Open MySQL CLI or MySQL Workbench and run:

```
CREATE DATABASE mydb CHARACTER SET UTF8MB4 COLLATE utf8mb4_general_ci;
CREATE USER 'myuser'@'localhost' IDENTIFIED BY 'mypassword';
GRANT ALL PRIVILEGES ON mydb.* TO 'myuser'@'localhost';
FLUSH PRIVILEGES;
```

☑ Step 4: Update `settings.py`

Edit `myproject/settings.py`:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'mydb',
        'USER': 'myuser',
        'PASSWORD': 'mypassword',
        'HOST': 'localhost',
        'PORT': '3306',
        'OPTIONS': {
            'init_command': "SET sql_mode='STRICT_TRANS_TABLES'",
        }
    }
}
```

☑ Step 5: Apply Migrations

Run:

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

If successful, Django will create tables in your MySQL database.

☑ Step 6: Test Connection

Run the development server:

```
python manage.py runserver
```

No errors? ✅ You're connected to MySQL!

☑ Optional: Create Superuser for Admin Panel

```
python manage.py createsuperuser
```

Tips

Topic	Description
`mysqlclient`	Fast and preferred adapter (C-based)
`pymysql`	Pure Python, easier to install, slightly slower
`MySQL Workbench`	GUI to view your database tables
`init_command`	Ensures Django works well with MySQL strict mode

Excellent, Maheshwaran! Let's move into **Day 6: Django Admin Panel** — one of Django's most powerful built-in features. The **Admin Panel** lets you manage models (database data) without writing code for a UI.

Django Admin Panel

1. ✅ Create a Superuser
2. ✅ Register Models in `admin.py`
3. ✅ Customize Admin Display
4. ✅ Add Filters and Search in Admin

1. Create Superuser

To log in to the Django admin panel, you need a **superuser account**.
`python manage.py createsuperuser`

🔤 You will be prompted to enter:

- Username
- Email
- Password (twice)

Once created, you can log in to the admin panel at:

🔗 http://127.0.0.1:8000/admin/

☑ 2. Register Models in Admin

By default, models won't show up in the admin panel until you **register them**.

Example: Register a `Product` model

In `myapp/admin.py`:

```
from django.contrib import admin
from .models import Product

admin.site.register(Product)
```

Now `Product` will appear in the admin sidebar.

☑ 3. Customize Admin Display

You can customize how the model is shown in the list view by creating a `ModelAdmin` class.

Example:

```
class ProductAdmin(admin.ModelAdmin):
    list_display = ('id', 'name', 'price', 'in_stock')  # Columns to
show
    list_editable = ('price', 'in_stock')  # Make editable directly
    ordering = ('-id',)  # Sort by ID descending

admin.site.register(Product, ProductAdmin)
```

☑ 4. Filters and Search in Admin

🔍 Add a search bar

```
class ProductAdmin(admin.ModelAdmin):
    search_fields = ('name', 'description')
```

This allows admin users to search `name` and `description`.

☑ Add filters

```
class ProductAdmin(admin.ModelAdmin):
    list_filter = ('in_stock',)
```

This adds a filter panel to the right for the `in_stock` field.

☑ Full Example: admin.py

```
from django.contrib import admin
from .models import Product

class ProductAdmin(admin.ModelAdmin):
    list_display = ('id', 'name', 'price', 'in_stock')
    list_editable = ('price', 'in_stock')
    search_fields = ('name', 'description')
    list_filter = ('in_stock',)
    ordering = ('-id',)

admin.site.register(Product, ProductAdmin)
```

☑ Summary

Feature	Syntax Example
Create superuser	`python manage.py createsuperuser`
Register model	`admin.site.register(Model)`

List display columns	`list_display = ('field1', 'field2')`
Search fields	`search_fields = ('field1',)`
List filters	`list_filter = ('field',)`
Editable in list view	`list_editable = ('field',)`
Ordering	`ordering = ('-id',)`

Day 77 Tasks

Django ORM, Models, Database, and Admin Panel: 50 Practice Tasks

ORM & Models (Python Classes to Database Tables)

1. Explain in your own words what ORM is and why it's useful.
2. Create a new Django app called `store`.
3. Define a `Product` model in `store/models.py` with fields: name, price, description, in_stock.
4. Add `__str__` method to your Product model to return the product's name.
5. Add the new app to `INSTALLED_APPS` in `settings.py`.
6. Define a `Category` model and link it to `Product` using a `ForeignKey`.
7. Add a `created_at` `DateTimeField` to the Product model with `auto_now_add=True`.
8. Add a `SKU` field as a unique identifier for each Product.
9. Add a `discount` field as a DecimalField (nullable).
10. Document the difference between `CharField`, `TextField`, `BooleanField`, and `DecimalField`.

Migrations

11. Run `python manage.py makemigrations` to create migrations for your app.
12. Inspect the generated migration file under `store/migrations/`.
13. Run `python manage.py migrate` to apply migrations and create database tables.
14. Roll back a migration using `python manage.py migrate store 0001`.
15. Add a new field (e.g., `brand`) to your Product model and make a new migration.
16. Remove the field, migrate again, and observe database changes.
17. Generate a blank migration with `python manage.py makemigrations --empty store`.
18. Use the Django shell (`python manage.py shell`) to create a Product instance.

19. Use the shell to query all products with `Product.objects.all()`.
20. Update a Product's price using the ORM and save the change.

SQLite (Default Database)

21. Locate and inspect the `db.sqlite3` file in your project.
22. Install `DB Browser for SQLite` and open the `db.sqlite3` file.
23. View the Product table and confirm columns match your model.
24. Use Django shell to filter products with `in_stock=True`.
25. Delete a Product record using the ORM.

MySQL Integration

26. Install MySQL server on your system.
27. Install `mysqlclient` or `pymysql` in your virtual environment.
28. Run `import pymysql; pymysql.install_as_MySQLdb()` in your project's `__init__.py` if using PyMySQL.
29. Create a new MySQL database and user using MySQL CLI or Workbench.
30. Update the `DATABASES` setting in `settings.py` to use MySQL.
31. Run `python manage.py migrate` to create tables in MySQL.
32. Use MySQL Workbench to view Django-created tables.
33. Create a superuser and log into the Django admin using MySQL as the backend.
34. Explain the difference between SQLite and MySQL in Django projects.

Django Admin Panel

35. Run `python manage.py createsuperuser` and set up an admin account.
36. Register the Product model in `admin.py`.
37. Log into the admin panel at `/admin/` and add a Product entry.
38. Register the Category model in `admin.py` and add a Category.
39. Assign a Product to a Category via the admin form.
40. Unregister a model from the admin and confirm it disappears from the admin site.

Admin Customizations

41. Create a `ProductAdmin` class to customize Product display.
42. Set `list_display` to show `id`, `name`, `price`, `in_stock` in the admin list.
43. Add `list_editable` for the `price` and `in_stock` fields.

44. Add `search_fields` for `name` and `description`.
45. Add `list_filter` for the `in_stock` field.
46. Set `ordering` to `-id` so newest products come first.
47. Register Product using `admin.site.register(Product, ProductAdmin)`.
48. Test the search bar and filter sidebar in the admin panel.
49. Make a Product field read-only in the admin form (e.g., `created_at`).
50. Document all your admin customizations with comments in `admin.py`.

Day 77 Mini Project Requirements

1. Product Inventory Manager

- **Models:** Product (name, SKU, price, description, in_stock, created_at), Category (name)
- **Features:** Products belong to categories. Use Boolean in_stock, DecimalField price, etc.
- **Admin:** Register models, show list_display (id, name, price, in_stock), list_filter (category, in_stock), search_fields (name, description), editable price/in_stock.
- **Database:** SQLite or MySQL.
- **Task:** Manage inventory from admin (add/edit/delete, filter, search).

2. Library Book Catalog

- **Models:** Book (title, author, isbn, published_date, is_available), Genre (name)
- **Features:** Book foreign key to Genre, admin can manage book listings.
- **Admin:** Customize list_display, add search on title/author, filter by genre/is_available.
- **Task:** Admin can add books, assign to genres, mark as available/unavailable.

3. Employee Directory

- **Models:** Employee (first_name, last_name, email, department, hire_date, is_active)
- **Features:** Department as a CharField/choice; is_active Boolean.

- **Admin:** List view shows active/inactive, filter/search by department or name, edit is_active inline.
- **Task:** Use admin to manage employee roster.

4. Simple Blog Manager

- **Models:** Post (title, body, published, created_at), Tag (name)
- **Features:** ManyToMany between Post and Tag; published Boolean.
- **Admin:** Search/filter by tag, published, date; list_editable for published.
- **Task:** Admin curates posts/tags via admin panel.

5. Student Enrollment Tracker

- **Models:** Student (name, age, email, enrolled_on, is_active), Course (name, credits)
- **Features:** Student can enroll in multiple courses (ManyToMany).
- **Admin:** Filter/search students by course/enrollment; editable is_active.
- **Task:** Track enrollments and manage student records.

6. Customer Feedback Collector

- **Models:** Feedback (customer_name, email, message, submitted_at, reviewed)
- **Features:** reviewed Boolean, message as TextField.
- **Admin:** Filter/search by reviewed status, email; list_display feedback.
- **Task:** Admin reviews, marks feedback as reviewed.

7. Event Registration System

- **Models:** Event (title, date, location, is_open), Participant (name, email, registered_on, event)
- **Features:** Participant foreign key to Event; is_open Boolean for event.
- **Admin:** Filter/search by event, participant, is_open; inline edit is_open.
- **Task:** Manage events and participant signups.

8. Online Store Category/Product Manager

- **Models:** Category (name), Product (name, price, category, in_stock, created_at)
- **Features:** Product linked to Category; price as DecimalField.
- **Admin:** Products grouped by category, filter by in_stock/category, search by name.
- **Task:** Manage store inventory and categories.

9. Vehicle Fleet Tracker

- **Models:** Vehicle (plate_number, model, type, is_available, purchased_on)
- **Features:** type as CharField/choice (car, van, truck), is_available Boolean.
- **Admin:** Filter/search by type, is_available; list_display all fields.
- **Task:** Track vehicles and their status.

10. School Subject/Class Assignments

- **Models:** Subject (name), Teacher (name, email, subject)
- **Features:** Teacher foreign key to Subject.
- **Admin:** Filter teachers by subject, search by name, edit email inline.
- **Task:** Manage subject-teacher assignments.

11. Company Asset Management

- **Models:** Asset (name, asset_tag, assigned_to, purchased_on, is_functional)
- **Features:** assigned_to as CharField (employee name), is_functional Boolean.
- **Admin:** Filter/search by assigned_to, is_functional.
- **Task:** Track assets and their assignments.

12. Cinema Movie Schedule

- **Models:** Movie (title, genre, duration), Screening (movie, show_time, screen_number, is_active)
- **Features:** Screening linked to Movie, is_active Boolean.
- **Admin:** Filter/search by movie/active status, edit show_time inline.
- **Task:** Manage movies and their showings.

13. Restaurant Menu Manager

- **Models:** MenuItem (name, price, category, is_available), Category (name)
- **Features:** MenuItem linked to Category; price as DecimalField.
- **Admin:** Filter by category, is_available; search by name.
- **Task:** Update menu from admin.

14. Appointment Scheduling System

- **Models:** Appointment (client_name, date, time, status, notes)
- **Features:** status as choices (scheduled, completed, cancelled).
- **Admin:** Filter/search by status/date, edit status inline.
- **Task:** Admin schedules and tracks appointments.

15. Job Application Tracker

- **Models:** Job (title, department, posted_on), Applicant (name, email, applied_on, job, status)
- **Features:** Applicant linked to Job; status as choices (pending, reviewed, hired, rejected).
- **Admin:** Filter/search by job/status, list_display applicant info.
- **Task:** Track applications and applicants.

16. School Attendance Log

- **Models:** Student (name, class_name), Attendance (student, date, present)
- **Features:** Attendance linked to Student; present Boolean.
- **Admin:** Filter by class, date, present; search by student.
- **Task:** Record and view attendance.

17. Clinic Patient Records

- **Models:** Patient (name, dob, contact), Visit (patient, visit_date, reason, doctor_name)

- **Features:** Visit linked to Patient.
- **Admin:** Filter/search by patient/doctor, visit_date.
- **Task:** Track visits and patients.

18. Sports Team Roster

- **Models:** Team (name, coach), Player (name, position, team, is_active)
- **Features:** Player foreign key to Team; is_active Boolean.
- **Admin:** Filter/search by team/position/active.
- **Task:** Manage teams and player assignments.

19. Membership Subscription System

- **Models:** Member (name, email, join_date, is_active), Subscription (member, plan_name, start_date, end_date, status)
- **Features:** Subscription linked to Member; status as choices (active, expired, cancelled).
- **Admin:** Filter/search by plan/status, edit status inline.
- **Task:** Track member subscriptions.

20. Pet Adoption Registry

- **Models:** Pet (name, type, age, adopted, intake_date), Adopter (name, contact, pet)
- **Features:** Adopter linked to Pet; adopted Boolean.
- **Admin:** Filter/search by type/adopted, list_display adopter/pet info.
- **Task:** Manage adoptable pets and adoptions.