TRAINING DAY6 REPORT

30 JUNE 2025

What is ORM?

ORM is a technique that maps classes in code to tables in a database and maps class attributes to table columns. It allows for CRUD (Create, Read, Update, Delete) operations using object-oriented code.

In Django:

- Each model class maps to a database table.
- Each model field maps to a column.
- Each model instance maps to a row.

Django ORM Architecture

Django ORM sits between the model layer and the database. When you run Django commands or write queries in code, Django translates your Python logic into SQL queries internally.

Why Use Django ORM?

→ **Abstraction:** No need to write SQL manually.

→ **Productivity:** Speeds up development by handling repetitive database tasks.

→ **Security:** Protects against SQL injection.

→ **Database Portability:** You can switch between databases (SQLite, PostgreSQL, MySQL) with minimal code changes.

ORM Queries in Views or Shell:

```
(.venv) PS C:\Users\WELCOME\Desktop\django\blogapi> python manage.py shell
7 objects imported automatically (use -v 2 for details).

Python 3.13.2 (tags/v3.13.2:4f8bb39, Feb  4 2025, 15:23:48) [MSC v.1942 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>> from blog.models import Product
>>> Product.objects.create(name="Shoes")
<Product: Product object (2)>
>>> Product.objects.filter(name__icontains="shoe")
<QuerySet [<Product: Product object (1)>, <Product: Product object (2)>]>
>>> Product.objects.all()
<QuerySet [<Product: Product object (1)>, <Product: Product object (2)>]>
>>> exit()
now exiting InteractiveConsole...
```

Common ORM Operations

1. Create Records

```
(.venv) PS C:\Users\WELCOME\Desktop\django\blogapi> python manage.py shell
7 objects imported automatically (use -v 2 for details).

Python 3.13.2 (tags/v3.13.2:4f8bb39, Feb  4 2025, 15:23:48) [MSC v.1942 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>> from blog.models import Product
>>> Product.objects.create(name="Shoes")
<Product: Product object (2)>
```

2. Read Records

```
(.venv) PS C:\Users\WELCOME\Desktop\django\blogapi> python manage.py shell
7 objects imported automatically (use -v 2 for details).

Python 3.13.2 (tags/v3.13.2:4f8bb39, Feb  4 2025, 15:23:48) [MSC v.1942 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>> from blog.models import Product
>>> Product.objects.all()
<QuerySet [<Product: Product object (1)>, <Product: Product object (2)>]>
>>> Product.objects.get(id=1)
<Product: Product object (1)>
>>> Product.objects.values()
<QuerySet [{'id': 1, 'created_at': datetime.datetime(2025, 7, 10, 8, 43, 38, 480582, tzinfo=datetime.timezone.utc), 'name': 'Sports shoes'}, {'id': 2, '
created_at': datetime.datetime(2025, 7, 10, 8, 47, 55, 839487, tzinfo=datetime.timezone.utc), 'name': 'Shoes'}]>
>>> exit()
now exiting InteractiveConsole...
```

3. Update Records

```
(.venv) PS C:\Users\WELCOME\Desktop\django\blogapi> python manage.py shell
7 objects imported automatically (use -v 2 for details).

Python 3.13.2 (tags/v3.13.2:4f8bb39, Feb  4 2025, 15:23:48) [MSC v.1942 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>> from blog.models import Product
>>> p = Product.objects.get(id=1)
>>> p.name = "Sports shoes"
>>> p.save()
>>> exit()
now exiting InteractiveConsole...
```

4. Delete Records

```
(.venv) PS C:\Users\WELCOME\Desktop\django\blogapi> python manage.py shell
7 objects imported automatically (use -v 2 for details).

Python 3.13.2 (tags/v3.13.2:4f8bb39, Feb  4 2025, 15:23:48) [MSC v.1942 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>> from blog.models import Product
>>> Product.objects.get(id=1).delete()
(1, {'blog.Product': 1})
>>> exit()
now exiting InteractiveConsole...
```

Introduction to Migrations

Migrations are Django's way of updating the database schema in response to changes made in the model definitions. Instead of manually altering the database using SQL commands, migrations automate the process, keeping the database schema in sync with your Python code.

Why Are Migrations Important?

Migrations offer several advantages:

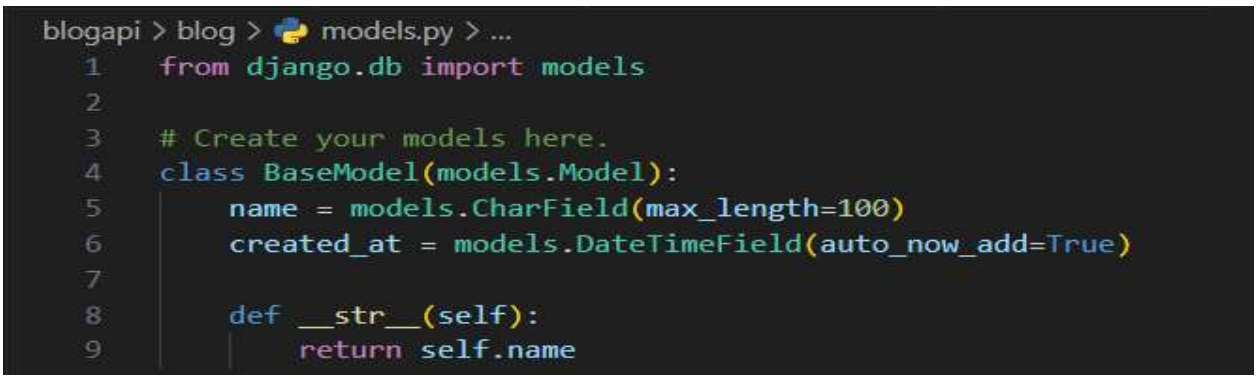
1. Automatic Schema Updates: Reflect changes in models without touching SQL.
2. Version Control: Every schema change is stored in a migration file that can be tracked in Git.

3. Portability: Enables easy deployment on production or other environments.
4. Rollback Support: Allows undoing changes using reverse migrations.

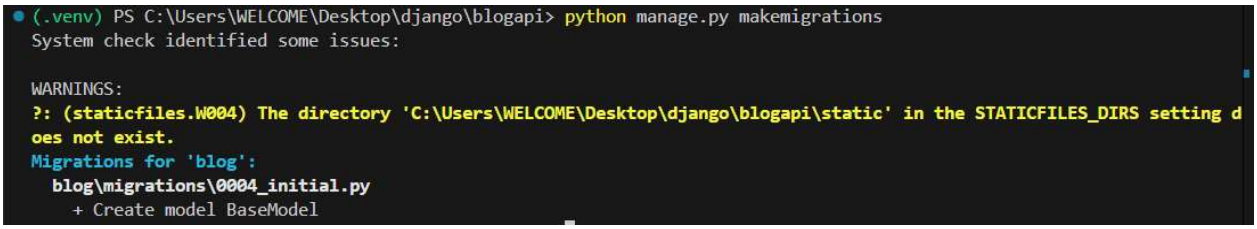
Migration Lifecycle and Workflow

The typical migration workflow consists of four main steps:

1. Making Changes to Models:



2. Creating Migrations:



This command checks for any changes in your models and generates a migration file (like 0001_initial.py) inside the app's migrations/ folder.

What it does:

- Tracks additions, deletions, or modifications to models.
- Prepares Python files that describe how to apply these changes to your database.

3. Applying Migrations:

To apply these migration files to your actual database, run:

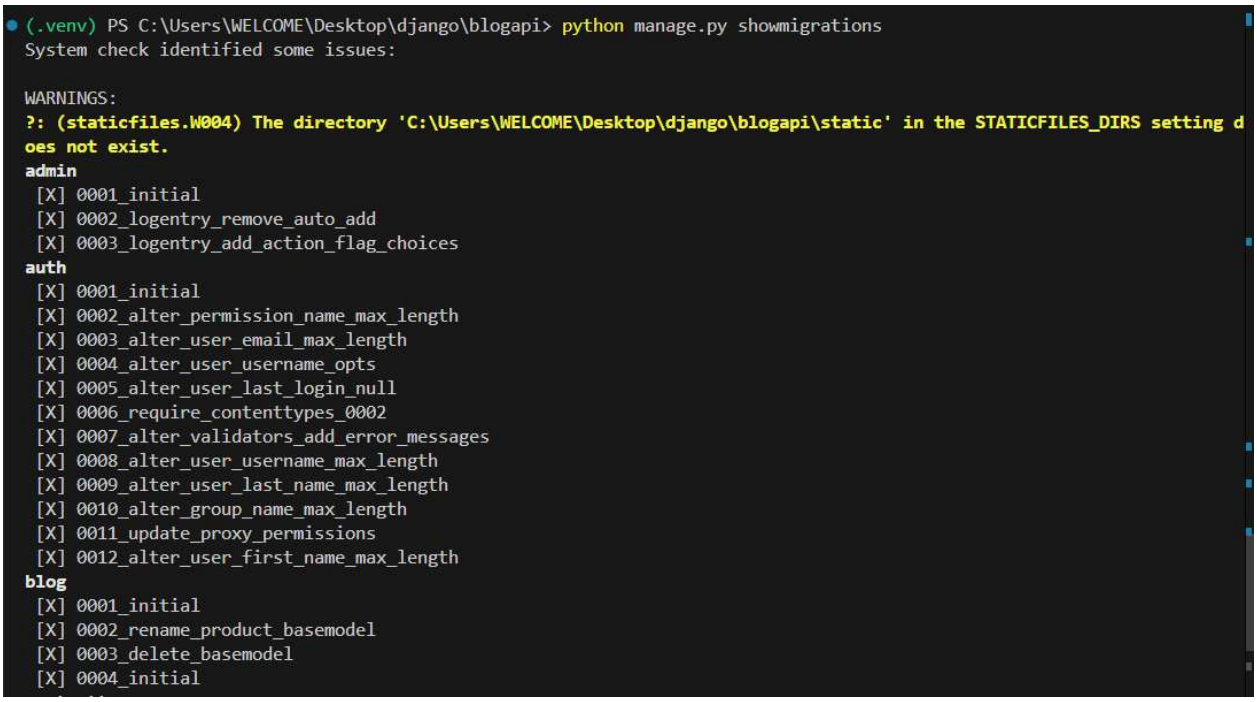
```
(.venv) PS C:\Users\WELCOME\Desktop\django\blogapi> python manage.py migrate
System check identified some issues:

WARNINGS:
?: (staticfiles.W004) The directory 'C:\Users\WELCOME\Desktop\django\blogapi\static' in the STATICFILES_DIRS setting d
oes not exist.
Operations to perform:
  Apply all migrations: admin, auth, blog, contenttypes, sessions
Running migrations:
  Applying blog.0002_rename_product_basemodel... OK
  Applying blog.0003_delete_basemodel... OK
  Applying blog.0004_initial... OK
```

What it does:

- Executes SQL statements behind the scenes to match the database schema with your models.
- Also runs built-in migrations like for Django's auth system (auth, admin, sessions).

4. Checking Migration Status:

To see unapplied or applied migrations:



USEFUL MIGRATION COMMANDS

COMMAND	DESCRIPTION
makemigrations	Creates new migration files based on model changes
migrate	Applies the migrations to the actual database
showmigrations	Lists all migration files and shows applied/unapplied ones
sqlmigrate app_name migration_name	Displays the raw SQL that will be executed
migrate app_name zero	Rolls back all migrations of the given app (dangerous)