Tops Technology

Module 16)
Python DB and Framework

Presented By : Nandni Vala

Django Database Connectivity (MySQL or SQLite)

1.Connecting Django to a database (SQLite or MySQL)

- **Configure the Database in settings.py**
- **For SQLite (Default)**:
- SQLite is Django's default database, and it requires no additional installation. Ensure your settings.py contains:
- DATABASES = {
- 'default': {
- 'ENGINE': 'django.db.backends.sqlite3',
- 'NAME': BASE_DIR / 'db.sqlite3', # Database file path
- }
- }
- **For MySQL**:
- **Install MySQL Driver**: Install the `mysqlclient` package using pip:

- pip install mysqlclient
- **Update Database Settings**: Modify the DATABASES section in settings.py:
- DATABASES = {
- 'default': {
- 'ENGINE': 'django.db.backends.mysql',
- 'NAME': 'your_database_name',
- 'USER': 'your_username',
- 'PASSWORD': 'your_password',
- 'HOST': 'localhost', # Or the database server's IP address
- 'PORT': '3306',   # Default MySQL port
- }
- }
- **Create the Database (For MySQL Only)**
- Before running the project, create the database in MySQL:
- CREATE DATABASE your_database_name;

- **Apply Database Migrations**
- Run the following commands to create the necessary database tables:
- python manage.py makemigrations
- python manage.py migrate
- **Test the Connection**
- You can verify the database connection by running:
- python manage.py showmigrations

2.Using the Django ORM for database queries.

- he **Django ORM (Object-Relational Mapping)** provides an intuitive and Pythonic way to interact with the database by representing database tables as Python classes (models).
- **Basic ORM Queries**
- **1. Create Records**
- Use the create() method or instantiate and save a model instance.
- # Method 1: Using create()
- book = Book.objects.create(title="Django Basics", author="John Doe", price=9.99)
- # Method 2: Instantiating and saving
- book = Book(title="Advanced Django", author="Jane Doe", price=14.99)
- book.save()
- **Read Records**
- Fetch records using the ORM's query methods.
- **All Records**:
- books = Book.objects.all()

- **Filter Records**:
- cheap_books = Book.objects.filter(price__lt=10) # Books with price < 10
- **Get a Single Record**:
- book = Book.objects.get(id=1) # Raises `DoesNotExist` if no match
- **Query with Ordering**:
- books = Book.objects.order_by('price') # Ascending order
- books = Book.objects.order_by('-price') # Descending order
- **Select Specific Fields**:titles = Book.objects.values('title') # Returns dictionaries
- **Update Records**
- Modify existing records and save the changes.
- **Single Record**:
- book = Book.objects.get(id=1)
- book.price = 12.99
- book.save()

- **Multiple Records**:
- Book.objects.filter(price__lt=10).update(price=10)
- **Delete Records**
- Delete records using the delete() method.
- **Single Record**:
- book = Book.objects.get(id=1)
- book.delete()
- **Multiple Records**:
- Book.objects.filter(price__lt=10).delete()
- **Advanced ORM Queries**
- **Chaining Queries**: Combine multiple query filters.
- books = Book.objects.filter(author="John Doe").exclude(price__gt=20)
- **Aggregations**: Use annotate() or aggregate() for calculations.
- from django.db.models import Avg, Max
- avg_price = Book.objects.aggregate(Avg('price'))