

**April Evan Jean Y. Elico**

**BSCpE – 2A**

## **Laboratory Activity No. 2:**

**Laboratory Activity No. 2:**

**Topic belongs to: Software Design and Database Systems**

**Title:** *Designing the Database Schema for the Library Management System*

---

**Introduction:** In this activity, you will design the database schema for the Library Management System. The database will include tables for books, authors, users, and borrowing records. You will also learn how to use Django's ORM (Object-Relational Mapping) to define the models.

---

### **Objectives:**

- Design the database schema for the Library Management System.
  - Create Django models to represent the schema.
  - Use Django's ORM to interact with the database.
- 

**Theory and Detailed Discussion:** Django uses an ORM (Object-Relational Mapping) system to map Python objects to database tables. By defining models in Python code, Django automatically creates the corresponding database tables. We will start by designing the database schema with the necessary relationships between entities like books, authors, and users.

---

### **Materials, Software, and Libraries:**

- **Django** framework
  - **SQLite** database (default in Django)
-

**Time Frame:** 2 Hours

---

## Procedure:

### 1. Create Django Apps:

- In Django, an app is a module that handles a specific functionality. To keep things modular, we will create two apps: one for managing books and another for managing users.

```
python manage.py startapp books
python manage.py startapp users
```

```
[05/Feb/2025 14:20:57] "GET / HTTP/1.1" 200 12068
(library_env) PS C:\Users\EB204_U03\Desktop\ELICO\library_system> cd..
(library_env) PS C:\Users\EB204_U03\Desktop\ELICO> cd library_root
(library_env) PS C:\Users\EB204_U03\Desktop\ELICO\library_root> python manage.py startapp books
(library_env) PS C:\Users\EB204_U03\Desktop\ELICO\library_root> python manage.py startapp users
(library_env) PS C:\Users\EB204_U03\Desktop\ELICO\library_root> 
```

### 2. Define Models for the Books App:

- Open the books/models.py file and define the following models:

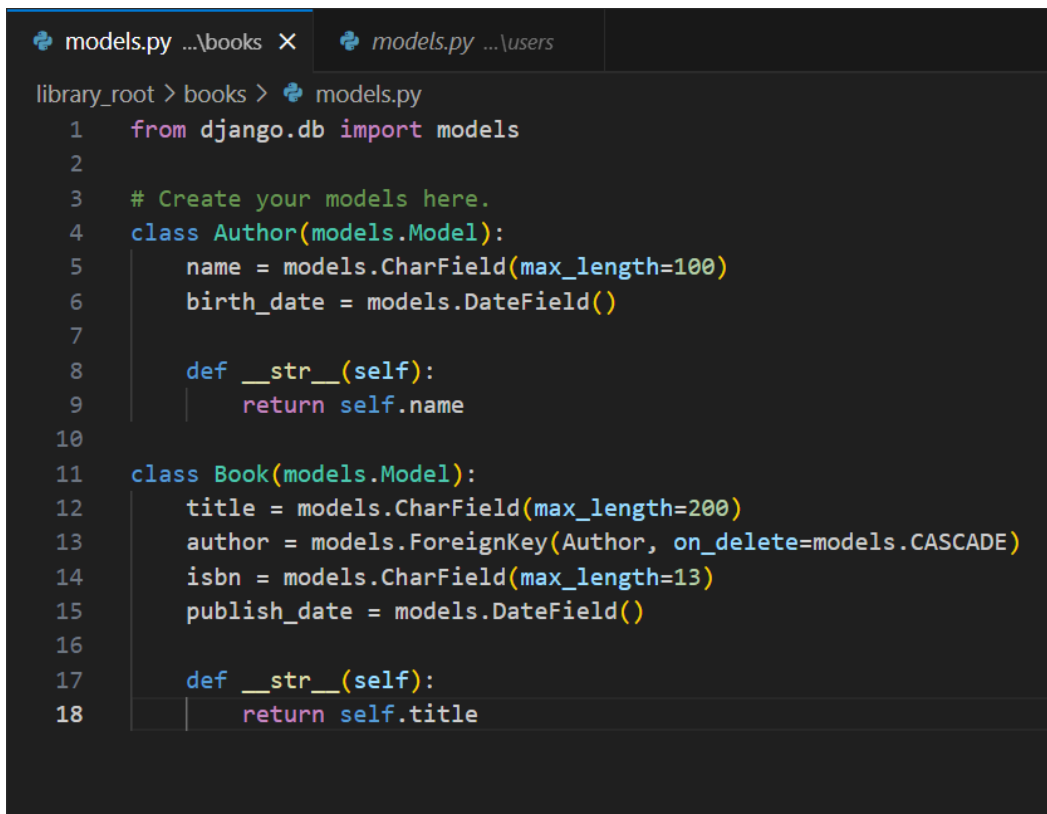
```
from django.db import models

class Author(models.Model):
    name = models.CharField(max_length=100)
    birth_date = models.DateField()

    def __str__(self):
        return self.name

class Book(models.Model):
    title = models.CharField(max_length=200)
    author = models.ForeignKey(Author, on_delete=models.CASCADE)
    isbn = models.CharField(max_length=13)
    publish_date = models.DateField()

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



```
models.py ...\books X  models.py ...\users

library_root > books > models.py
1  from django.db import models
2
3  # Create your models here.
4  class Author(models.Model):
5      name = models.CharField(max_length=100)
6      birth_date = models.DateField()
7
8      def __str__(self):
9          return self.name
10
11 class Book(models.Model):
12     title = models.CharField(max_length=200)
13     author = models.ForeignKey(Author, on_delete=models.CASCADE)
14     isbn = models.CharField(max_length=13)
15     publish_date = models.DateField()
16
17     def __str__(self):
18         return self.title
```

### 3. Define Models for the Users App:

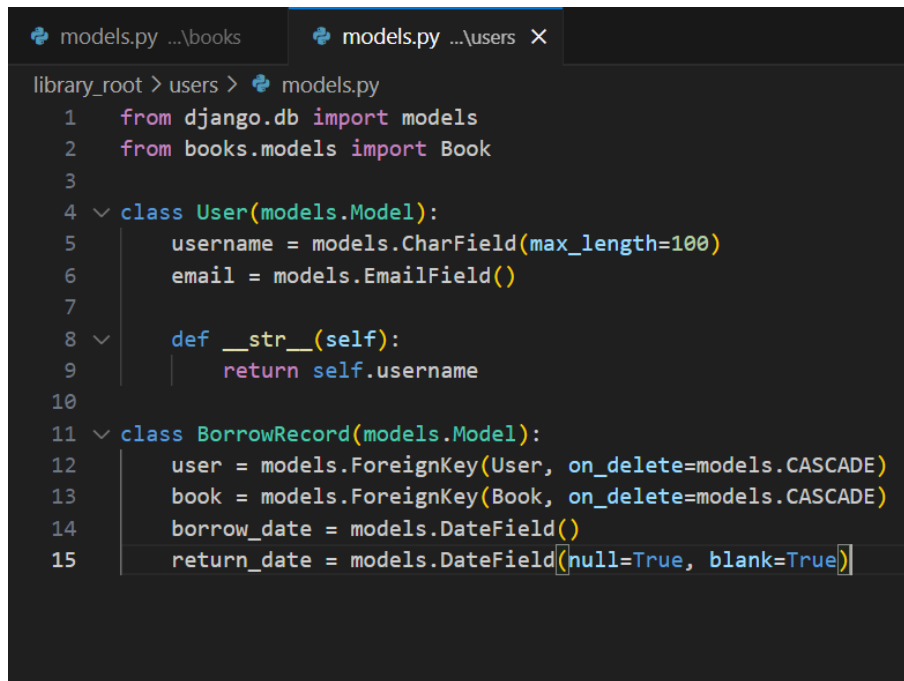
- Open the users/models.py file and define the following models:

```
from django.db import models
from books.models import Book

class User(models.Model):
    username = models.CharField(max_length=100)
    email = models.EmailField()

    def __str__(self):
        return self.username

class BorrowRecord(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    book = models.ForeignKey(Book, on_delete=models.CASCADE)
    borrow_date = models.DateField()
    return_date = models.DateField(null=True, blank=True)
```



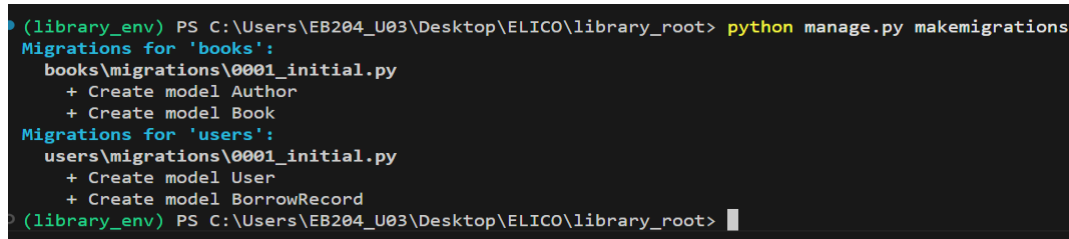
The screenshot shows a code editor with two tabs: 'models.py ...\books' and 'models.py ...\users'. The active tab is 'models.py ...\users', which contains the following Python code:

```
library_root > users > models.py
1  from django.db import models
2  from books.models import Book
3
4  class User(models.Model):
5      username = models.CharField(max_length=100)
6      email = models.EmailField()
7
8      def __str__(self):
9          return self.username
10
11 class BorrowRecord(models.Model):
12     user = models.ForeignKey(User, on_delete=models.CASCADE)
13     book = models.ForeignKey(Book, on_delete=models.CASCADE)
14     borrow_date = models.DateField()
15     return_date = models.DateField(null=True, blank=True)
```

#### 4. Apply Migrations:

- To create the database tables based on the models, run the following commands:

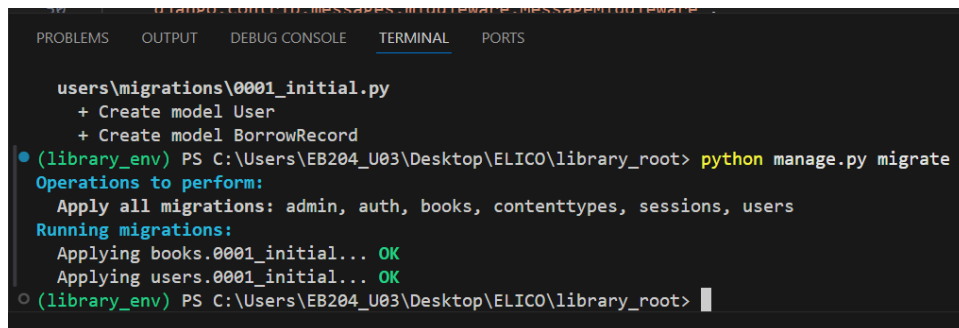
```
python manage.py makemigrations
```



The screenshot shows a terminal window with the following output:

```
(library_env) PS C:\Users\EB204_U03\Desktop\ELICO\library_root> python manage.py makemigrations
Migrations for 'books':
  books\migrations\0001_initial.py
    + Create model Author
    + Create model Book
Migrations for 'users':
  users\migrations\0001_initial.py
    + Create model User
    + Create model BorrowRecord
(library_env) PS C:\Users\EB204_U03\Desktop\ELICO\library_root>
```

```
python manage.py migrate
```



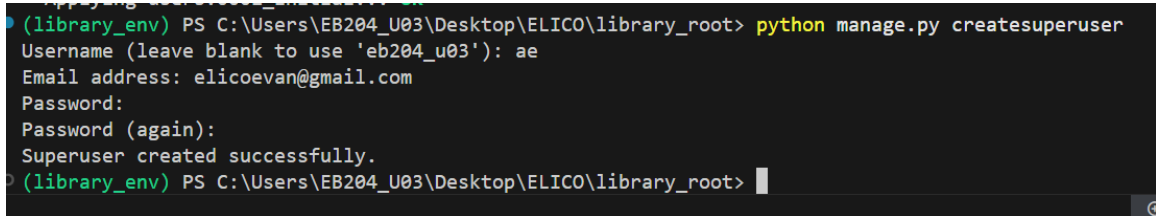
The screenshot shows a terminal window with the following output:

```
users\migrations\0001_initial.py
  + Create model User
  + Create model BorrowRecord
(library_env) PS C:\Users\EB204_U03\Desktop\ELICO\library_root> python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, books, contenttypes, sessions, users
Running migrations:
  Applying books.0001_initial... OK
  Applying users.0001_initial... OK
(library_env) PS C:\Users\EB204_U03\Desktop\ELICO\library_root>
```

## 5. Create Superuser for Admin Panel:

- Create a superuser to access the Django admin panel:

```
python manage.py createsuperuser
```



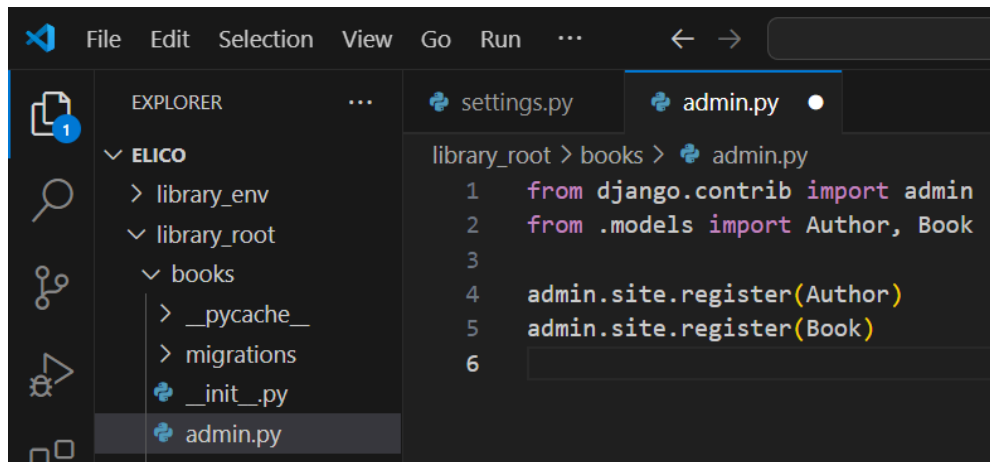
```
(library_env) PS C:\Users\EB204_U03\Desktop\ELICO\library_root> python manage.py createsuperuser
Username (leave blank to use 'eb204_u03'): ae
Email address: elicoevan@gmail.com
Password:
Password (again):
Superuser created successfully.
(library_env) PS C:\Users\EB204_U03\Desktop\ELICO\library_root>
```

## 6. Register Models in Admin Panel:

- In books/admin.py, register the Author and Book models:

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

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



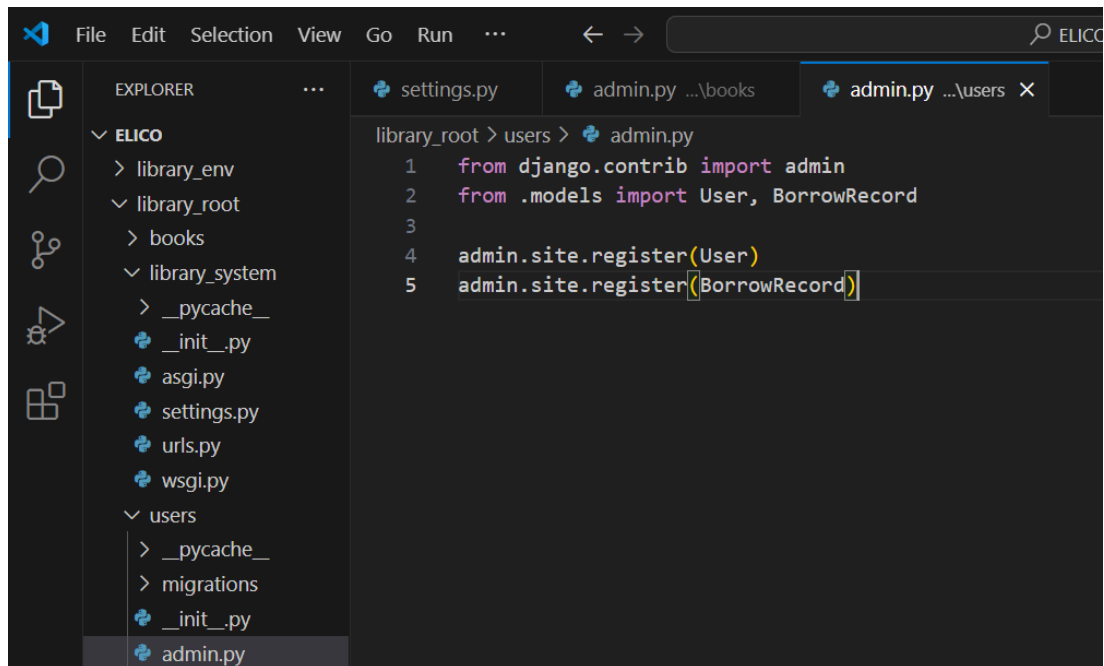
```
File Edit Selection View Go Run ...
EXPLORER
ELICO
  library_env
  library_root
    books
      __pycache__
      migrations
      _init_.py
      admin.py
  settings.py
  admin.py

library_root > books > admin.py
1 from django.contrib import admin
2 from .models import Author, Book
3
4 admin.site.register(Author)
5 admin.site.register(Book)
6
```

- In users/admin.py, register the User and BorrowRecord models:

```
from django.contrib import admin
from .models import User, BorrowRecord

admin.site.register(User)
admin.site.register(BorrowRecord)
```



## 7. Run the Development Server:

- Start the server again to access the Django admin panel:

```
python manage.py runserver
```

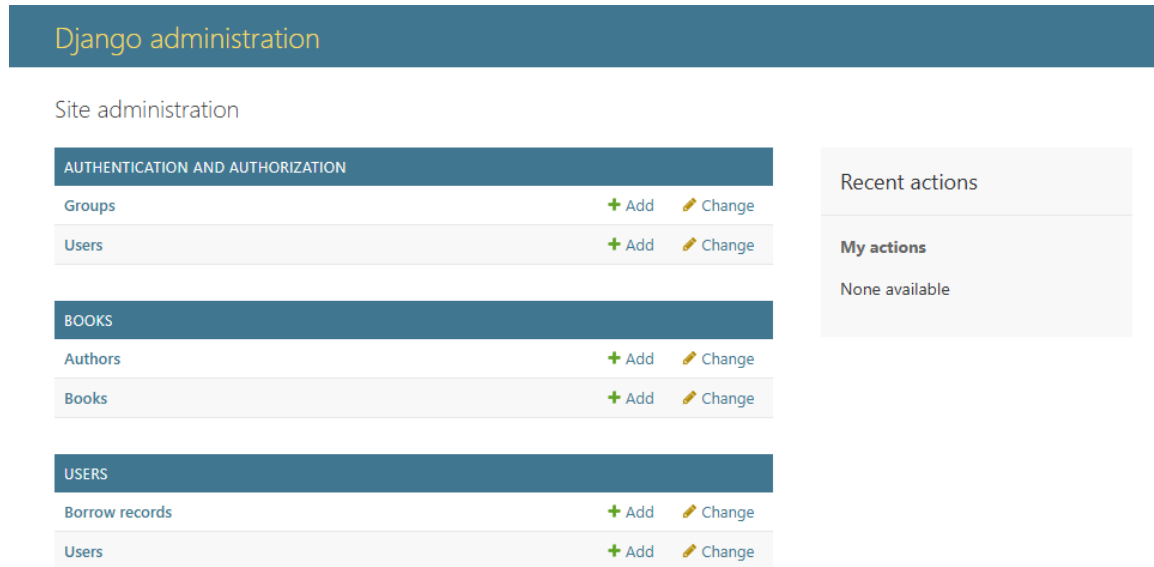
```
Superuser created successfully.
(library_env) PS C:\Users\EB204_U03\Desktop\ELICO\library_root> python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
February 05, 2025 - 15:26:20
Django version 5.1.5, using settings 'library_system.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.

[05/Feb/2025 15:26:28] "GET / HTTP/1.1" 200 12068
[05/Feb/2025 15:26:41] "GET /admin/ HTTP/1.1" 302 0
[05/Feb/2025 15:26:41] "GET /admin/login/?next=/admin/ HTTP/1.1" 200 4160
[05/Feb/2025 15:26:41] "GET /static/admin/css/login.css HTTP/1.1" 200 951
[05/Feb/2025 15:26:41] "GET /static/admin/css/nav_sidebar.css HTTP/1.1" 200 2810
[05/Feb/2025 15:26:41] "GET /static/admin/css/base.css HTTP/1.1" 200 22092
[05/Feb/2025 15:26:41] "GET /static/admin/css/dark_mode.css HTTP/1.1" 200 2804
[05/Feb/2025 15:26:41] "GET /static/admin/css/responsive.css HTTP/1.1" 200 17972
[05/Feb/2025 15:26:41] "GET /static/admin/js/theme.js HTTP/1.1" 200 1653
[05/Feb/2025 15:26:41] "GET /static/admin/js/nav_sidebar.js HTTP/1.1" 200 3063
```

## 8. Access Admin Panel:

- Go to `http://127.0.0.1:8000/admin` and log in using the superuser credentials. You should see the Author, Book, User, and BorrowRecord models.



**Django Program or Code:** Write down the summary of the code for models that has been provided in this activity.

### 1. Code Creating Django Apps:

```
python manage.py startapp books
python manage.py startapp users
```

### 2. Defining Models for the Books App:

```
from django.db import models

class Author(models.Model):
    name = models.CharField(max_length=100)
    birth_date = models.DateField()

    def __str__(self):
        return self.name

class Book(models.Model):
    title = models.CharField(max_length=200)
    author = models.ForeignKey(Author, on_delete=models.CASCADE)
    isbn = models.CharField(max_length=13)
    publish_date = models.DateField()
```

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

### 3. Defining Models for the Users App:

```
from django.db import models  
from books.models import Book  
  
class User(models.Model):  
    username = models.CharField(max_length=100)  
    email = models.EmailField()  
  
    def __str__(self):  
        return self.username  
  
class BorrowRecord(models.Model):  
    user = models.ForeignKey(User, on_delete=models.CASCADE)  
    book = models.ForeignKey(Book, on_delete=models.CASCADE)  
    borrow_date = models.DateField()  
    return_date = models.DateField(null=True, blank=True)
```

### 4. Applying Migrations:

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

### 5. Creating Superuser for Admin Panel:

```
python manage.py createsuperuser
```

### 6. Register Models in Admin Panel

#### In Books:

```
from django.contrib import admin  
from .models import Author, Book  
  
admin.site.register(Author)  
admin.site.register(Book)
```

#### In Users:

```
from django.contrib import admin  
from .models import User, BorrowRecord  
  
admin.site.register(User)  
admin.site.register(BorrowRecord)
```

### 7. Running the Development Server:

```
python manage.py runserver
```

---



**Results:** By the end of this activity, you will have successfully defined the database schema using Django models, created the corresponding database tables, and registered the models in the admin panel. (print screen the result and provide the github link of your work)

Site administration

The screenshot shows the Django Admin interface. The main content area has three sections: 'AUTHENTICATION AND AUTHORIZATION' with 'Groups' and 'Users' (each with '+ Add' and 'Change' links), 'BOOKS' with 'Authors' and 'Books' (each with '+ Add' and 'Change' links), and 'USERS' with 'Borrow records' and 'Users' (each with '+ Add' and 'Change' links). The right sidebar shows 'Recent actions' and 'My actions', which lists several actions for 'Beyond the Horizon' (Book) and 'Ae' (Author).

---

### Follow-Up Questions:

1. What is the purpose of using ForeignKey in Django models?

Ans. In Django models, a ForeignKey is used to create many-to-one relationships between tables in a relational database. It connects data by referring to the primary key of another table.

2. How does Django's ORM simplify database interaction?

Ans. Django's ORM is like a translator between the users Python code and database. Instead of writing complicated database language like SQL, you can just use regular Python. Django then takes the Python instructions and turns them into the SQL code needed to interact to the database.

---

### Findings:

In Django, we have what we call the 'ForeignKey' which helps connect different database tables by creating a many-to-one relationship. This means that multiple record in one table can be linked to a single record in another. It is a useful way to structure data efficiently and maintain relationships between different parts of a web application. Django's ORM, on the other hand, makes working with databases easier by letting developers write queries in Python instead of

SQL. Instead of manually writing complex SQL commands, you can interact with the database using simple Python functions, which speeds up development and reduces errors.

---

### **Summary:**

Django's 'ForeignKey' is a tool for linking tables in a relational database, ensuring organized and structured data. At the same time, its ORM simplifies database interactions by translating Python code into SQL queries, making database management much easier for developers.

---

### **Conclusion:**

Overall, Django provides a powerful yet user-friendly way to handle databases. Using 'ForeignKey' keeps data well-connected, while the ORM allows developers to focus on building features instead of dealing with complicated SQL queries. These features make Django a great choice for web development.