Name: Mark Anthony R. Alegre Yr & Sec: BSCpE-2B

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

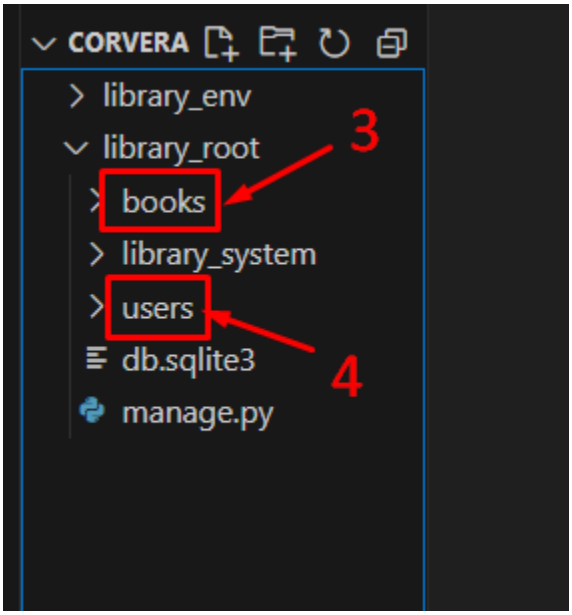
Procedure:

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.
    - Make sure that your are inside the *library_root* directory

python manage.py startapp books

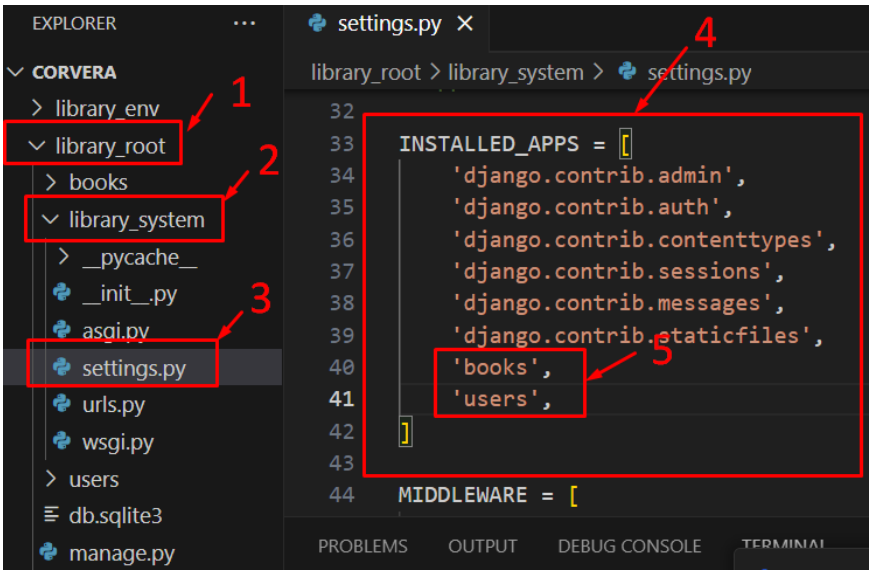
python manage.py startapp users

```
[05/Feb/2025 14:43:26] "GET / HTTP/1.1" 200 12068
(library_env) PS C:\Users\Admin\Desktop\CORVERA\library_root> python manage.py startapp books
(library_env) PS C:\Users\Admin\Desktop\CORVERA\library_root> python manage.py startapp users
(library_env) PS C:\Users\Admin\Desktop\CORVERA\library_root>
```



2. Register the Apps in Settings.py

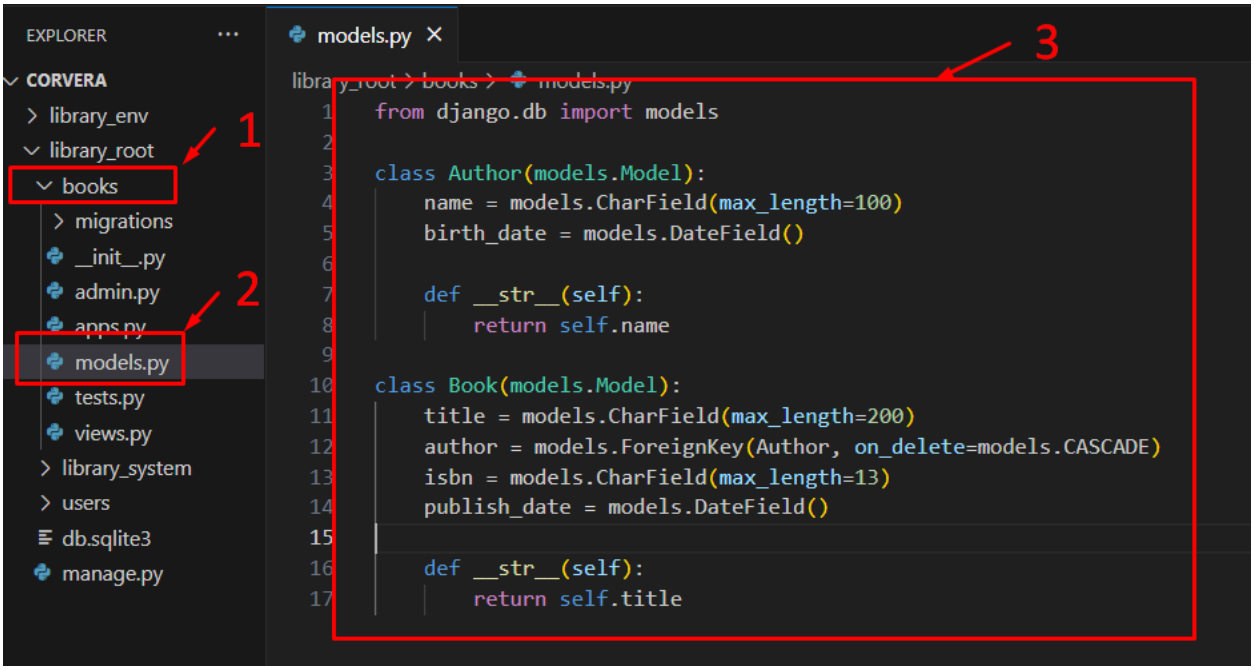
Under library_root directory open library_system>settings.py then add the books and users application under the INSTALLED_APPS



3. Define Models for the Books App:

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

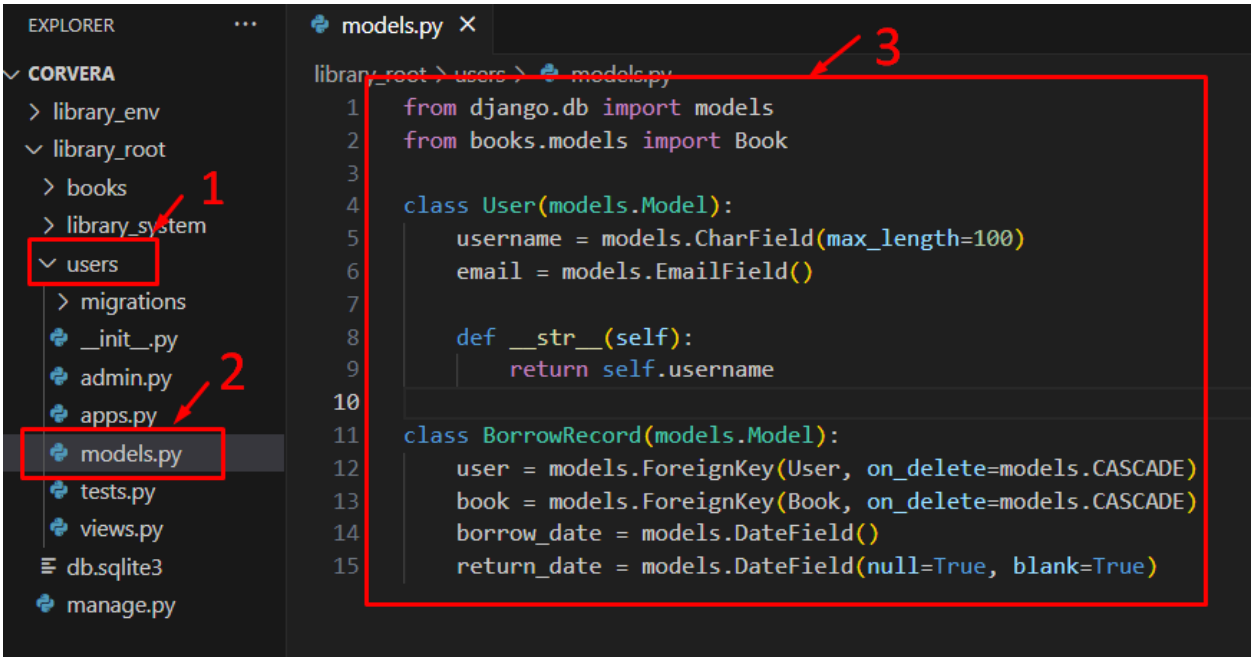
```
from django.db import models

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
```



2. **Define Models for the Users App**:
   - Open the users/models.py file and define the following models:

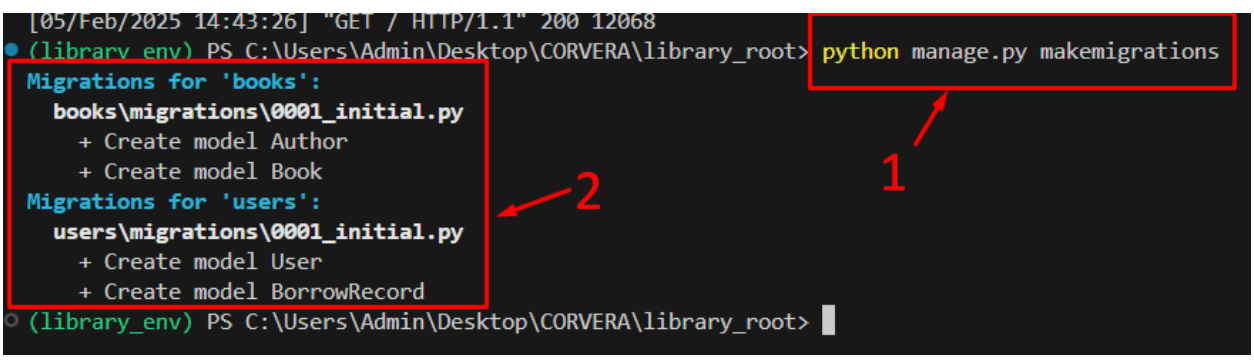
```
from django.db import models
from books.models import Book

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)



3. **Apply Migrations**:
   - To create the database tables based on the models, run the following commands:

python manage.py makemigrations



python manage.py migrate

```
(library_env) PS C:\Users\Admin\Desktop\CORVERA\library_root> python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, books, contenttypes, sessions, users
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying auth.0010_alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
  Applying auth.0012_alter_user_first_name_max_length... OK
  Applying books.0001_initial... OK
  Applying sessions.0001_initial... OK
  Applying users.0001_initial... OK
(library_env) PS C:\Users\Admin\Desktop\CORVERA\library_root>
```

1

2

4. **Create Superuser for Admin Panel**:
    - Create a superuser to access the Django admin panel:

python manage.py createsuperuser

Note: The password won't show when you type it.

```
(library_env) PS C:\Users\Admin\Desktop\CORVERA\library_root> python manage.py createsuperuser
Username (leave blank to use 'admin'):
Email address: admin@gmail.com
Password:
Password (again):
The password is too similar to the username.
This password is too short. It must contain at least 8 characters.
This password is too common.
Bypass password validation and create user anyway? [y/N]: y
Superuser created successfully.
(library_env) PS C:\Users\Admin\Desktop\CORVERA\library_root>
```

1

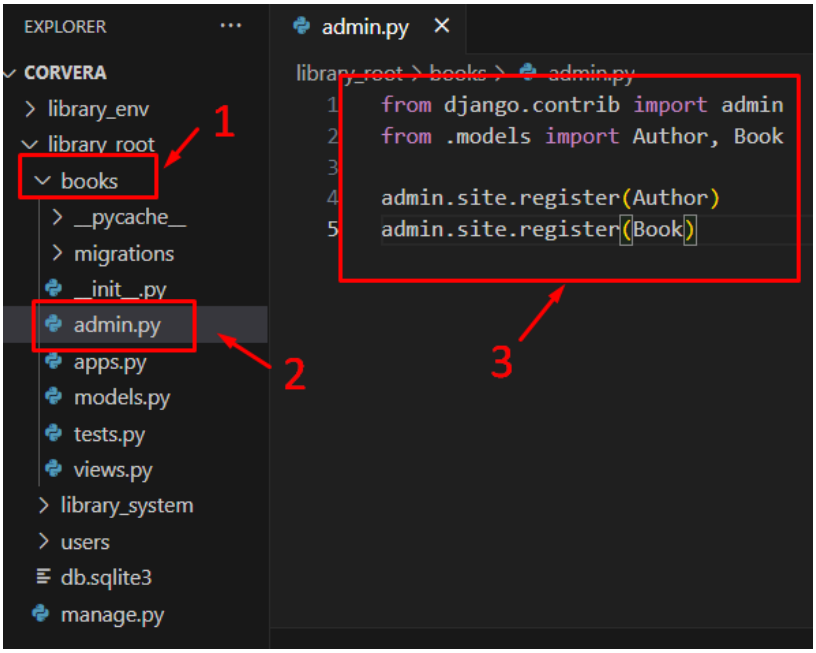
2

3

5. **Register Models in Admin Panel**:

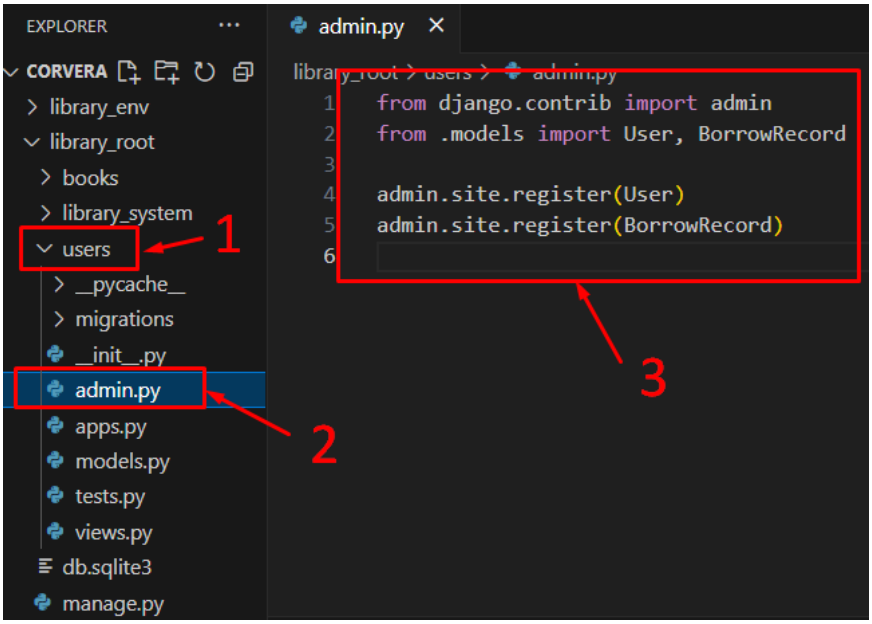
- In books/admin.py, register the Author and Book models:

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

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



- 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)
```

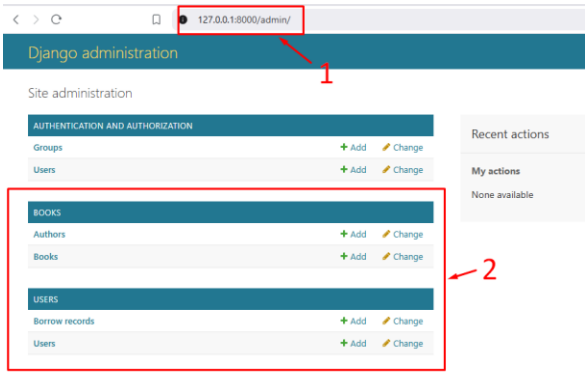


6. **Run the Development Server**:
    - Start the server again to access the Django admin panel:

python manage.py runserver

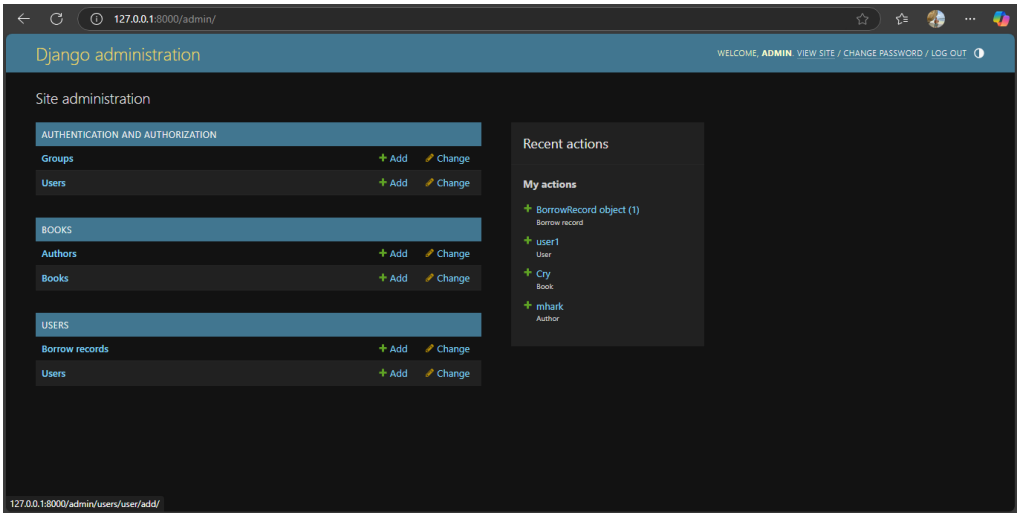
7. **Access Admin Panel**:

- Open a browser and 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.

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)



GitHub link: [mark-alegre01/Website at master]

Follow-Up Questions:

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

    - A ForeignKey in Django is used to link one model to another, creating a relationship between tables in the database. It's mainly used for many-to-one relationships, like when multiple orders belong to a single customer. This makes it easier to organize and retrieve related data efficiently.

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

    - Django's ORM (Object-Relational Mapper) lets you work with databases using Python instead of writing raw SQL queries. It handles tasks like inserting, updating, and retrieving data while ensuring security and efficiency. This makes database management much easier, especially for developers who don't want to deal with complex SQL syntax.

Findings:

- The ForeignKey field helps structure data properly by connecting related models, making queries and data management more straightforward.

- Django's ORM takes care of all the heavy lifting when working with databases, allowing developers to focus more on writing Python code instead of worrying about SQL.

Summary:

- Django makes working with databases easier by providing ForeignKey for relationships and an ORM that abstracts SQL queries. This saves time and effort while keeping everything clean and manageable.

Conclusion:

- With Django's ORM and ForeignKey, handling databases becomes much simpler. You don't need to write long SQL queries, and relationships between models are easy to manage. It's a huge advantage for developers looking for efficiency and ease of use.