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
```

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
```

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)
```

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

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

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

```
python manage.py createsuperuser
```

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)
```

   - 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:

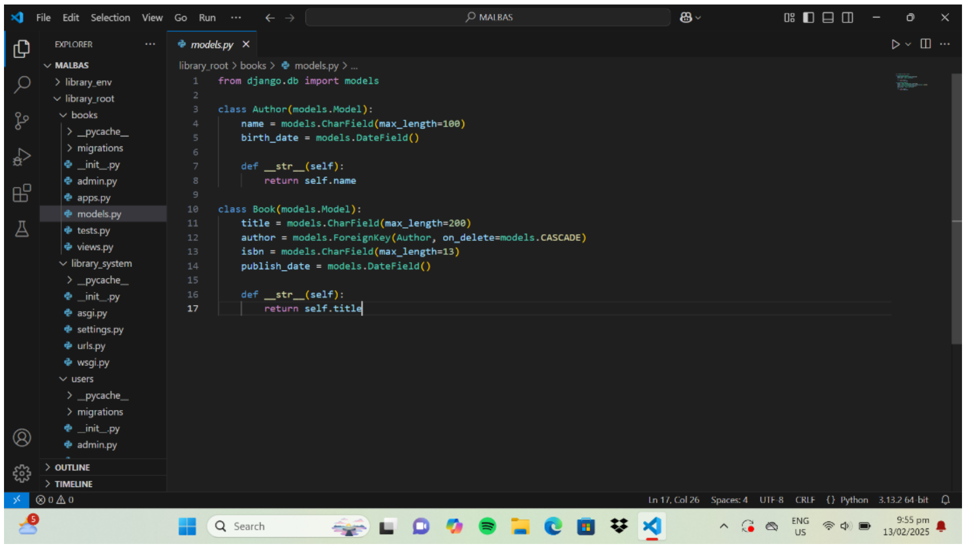
```
python manage.py runserver
```

8. **Access Admin Panel**:

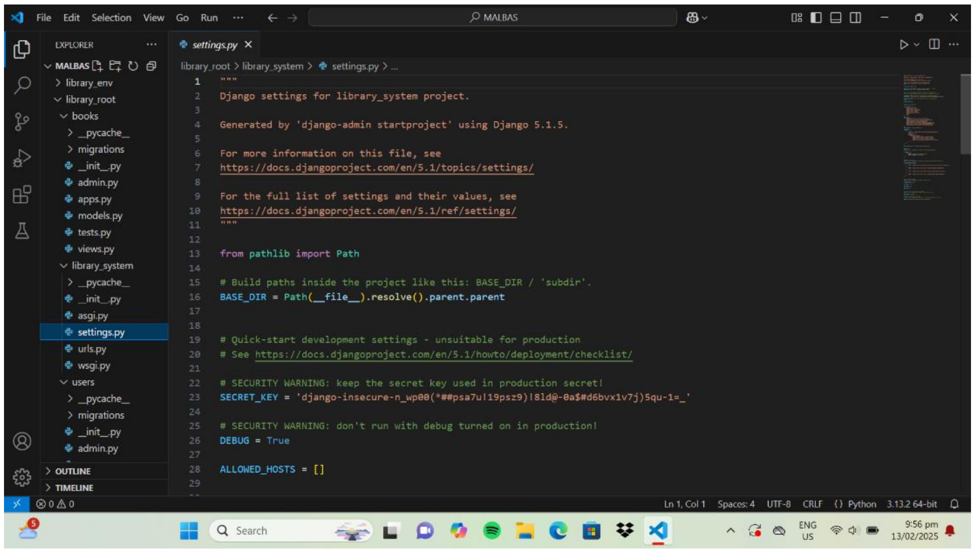
- Go to `http://127.0.0.1:8000/admin` and log in using the superuser credentials. You should see the `Author`, `Book`, `User`, and `BorrowRecord` models.

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

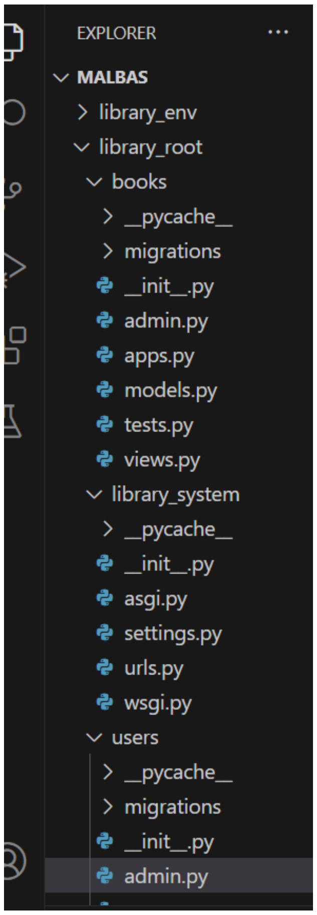
Results: By the end of this activity, you will have successfully defined the database schema using Django models, created the corresponding database tables, and registered the models in the admin panel. (print screen the result and provide the github link of your work)



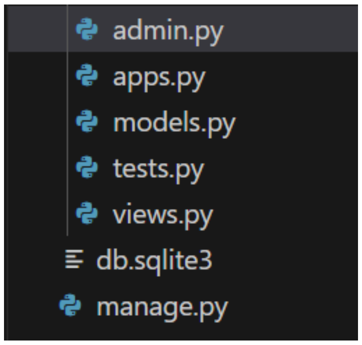
```python
from django.db import models

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
```



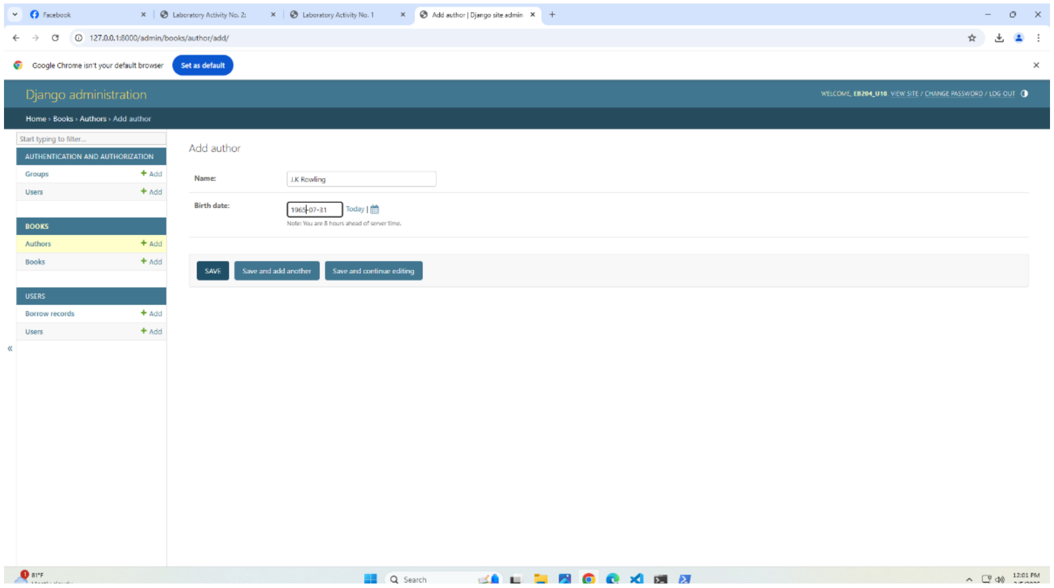
```python
"""
Django settings for library_system project.

Generated by 'django-admin startproject' using Django 5.1.5.

For more information on this file, see
https://docs.djangoproject.com/en/5.1/topics/settings/

For the full list of settings and their values, see
https://docs.djangoproject.com/en/5.1/ref/settings/
"""

from pathlib import Path

# Build paths inside the project like this: BASE_DIR / 'subdir'.
BASE_DIR = Path(__file__).resolve().parent.parent


# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/5.1/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = 'django-insecure-n_wp00(*##psa7u!19psz9)!8ld@-0a$#d6bvx1v7j)5qu-1=_'

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True

ALLOWED_HOSTS = []
```



```
EXPLORER ...
∨ MALBAS
  > library_env
  ∨ library_root
    ∨ books
      > __pycache__
      > migrations
      __init__.py
      admin.py
      apps.py
      models.py
      tests.py
      views.py
    ∨ library_system
      > __pycache__
      __init__.py
      asgi.py
      settings.py
      urls.py
      wsgi.py
    ∨ users
      > __pycache__
      > migrations
      __init__.py
      admin.py
```



```
    admin.py
    apps.py
    models.py
    tests.py
    views.py
  db.sqlite3
  manage.py
```



127.0.0.1:8000/admin/books/author/add/

Facebook
Laboratory Activity No. 2:
Laboratory Activity No. 1
Add author | Django site admin
Google Chrome isn't your default browser
Set as default
Django administration
WELCOME, EB204_U10. VIEW SITE / CHANGE PASSWORD / LOG OUT
Home › Books › Authors › Add author
Start typing to filter...
AUTHENTICATION AND AUTHORIZATION
Groups + Add
Users + Add
BOOKS
Authors + Add
Books + Add
USERS
Borrow records + Add
Users + Add

Add author

Name: J.K Rowling

Birth date: 1965-07-31 Today |

Note: You are 8 hours ahead of server time.

SAVE
Save and add another
Save and continue editing

Follow-Up Questions:

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

   To establish a many-to-one link between two Django models, a ForeignKey is utilized. It is a form of field that connects two models, enabling the establishment of connections in which an object from one related model can be connected to other objects in the current model. In Django, associations between models are created and managed using the ForeignKey, which promotes database integrity and makes related data querying easier.

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

   By supplying a Pythonic interface for database operations, abstracting away the complexity of SQL, granting database independence, and automating processes like validation and schema management, Django's ORM streamlines database interaction. This facilitates and expedites database development for developers.

Findings:

In Django models, a many-to-one link between two models is established using the ForeignKey field. It makes it possible for several instances of one model to be connected to one instance of another. The ForeignKey makes it simple to query related data without requiring complicated SQL expressions and guarantees database integrity by preventing problems like orphaned records. By enabling developers to use Python objects rather than raw SQL to carry out database operations, Django's Object-Relational Mapping (ORM) framework streamlines database interactions. In addition to abstracting the complexities of SQL and automating routine operations like validation, migrations, and schema management, the ORM offers a user-friendly, Pythonic interface for working with data. Because of its database independence, the same code can work with many databases, such as PostgreSQL, MySQL, or SQLite, without requiring any changes.

Summary:

In order to maintain integrity and streamline relevant data queries, ForeignKey establishes many-to-one links between models. A high-level, Pythonic interface for working with the database is offered by Django's ORM system, which manages tasks like data aggregation, ordering, and filtering. It simplifies database administration, abstracts the complexities of SQL, and automates processes like data validation and schema migrations.

Conclusion:

Django's ForeignKey is a strong tool for building connections across models, encouraging data consistency, and streamlining queries. Developers may work with databases more effectively and Pythonically when combined with Django's ORM, which streamlines and abstracts database operations. This makes Django an excellent choice for web development since it results in quicker development cycles, fewer errors, and better database administration.