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:

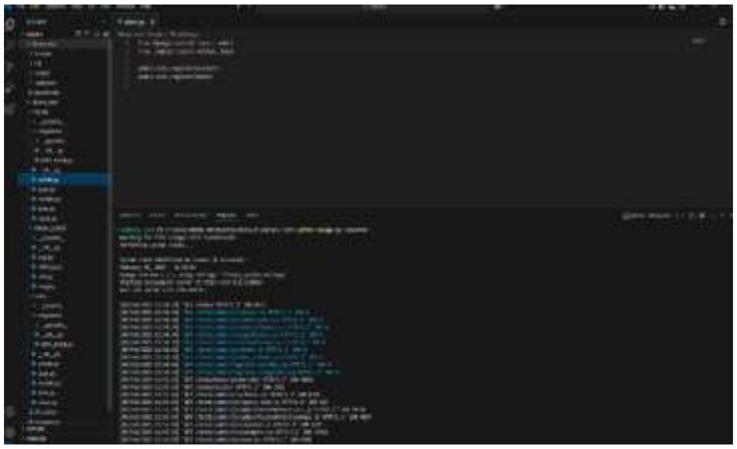
```
python manage.py runserver
```

8. **Access Admin Panel**:

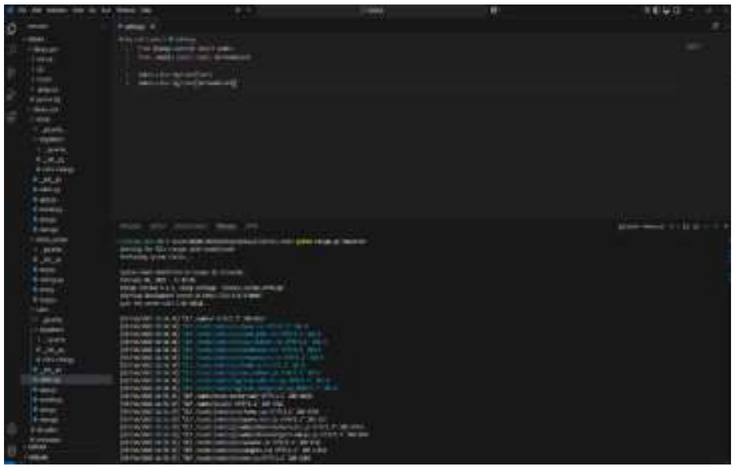
- Go to `http://127.0.0.1:8000/admin` and log in using the superuser credentials. You should see the `Author`, `Book`, `User`, and `BorrowRecord` models.

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)



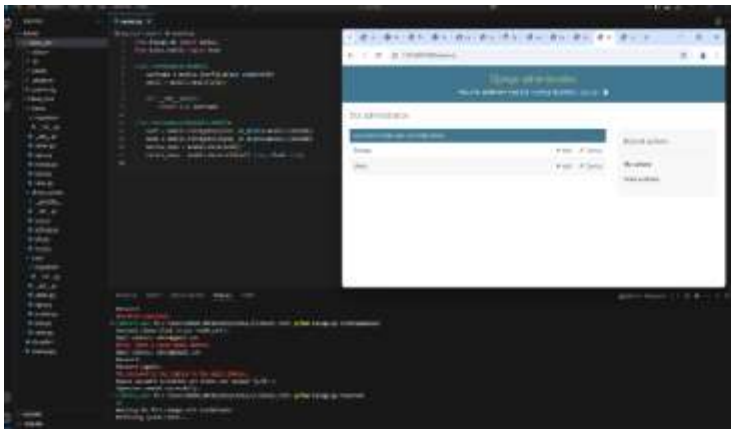
[illegible]



[illegible]

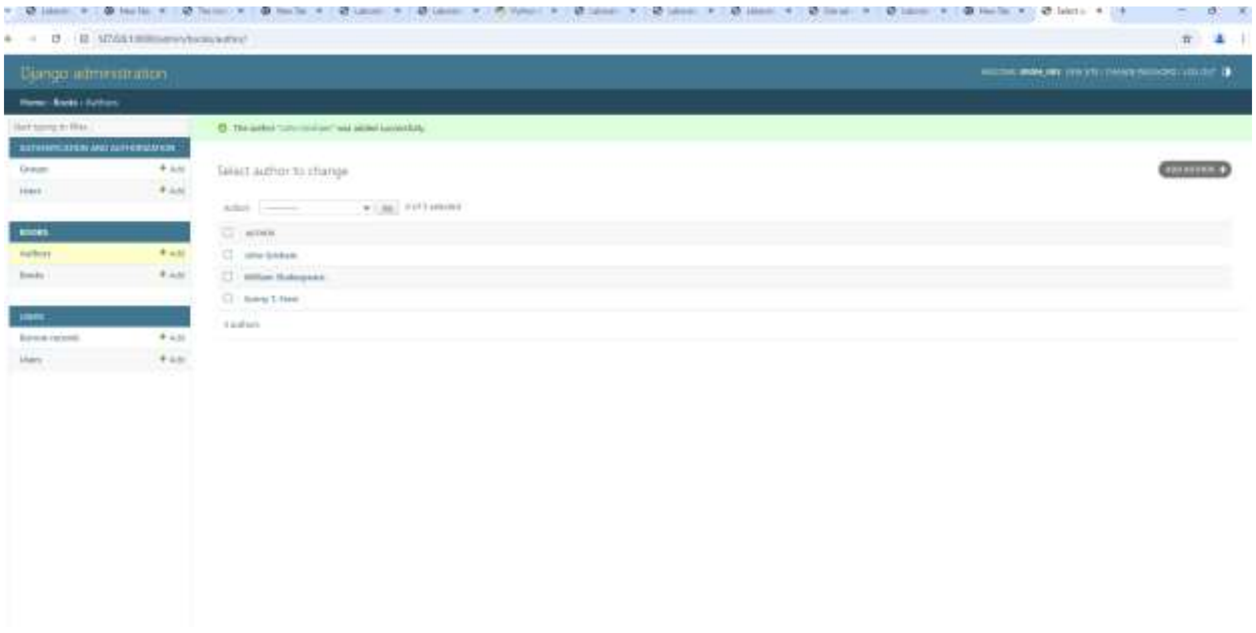
[illegible]

[illegible]



[illegible]

[illegible]


Django administration

Select author to change

William Shakespeare

Follow-Up Questions:

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

- In Django models, the use of ForeignKey is intended to construct a "many-to-one" link between two models.It enables you to link records across various database tables by allowing one model to reference another.In a blog application, for instance, a ForeignKey can be used to link each comment to a certain post, guaranteeing that each comment is unique to a single post even though a post may contain multiple comments.

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

- By enabling developers to communicate with the database using Python objects rather than performing raw SQL queries, Django's ORM (Object-Relational Mapping) streamlines database interaction.It makes database activities like generating, reading, updating, and removing entries easier by automatically translating Python code into SQL queries.This abstraction helps guarantee that database interactions are safe, effective, and less prone to errors by doing away with the need to write SQL by hand.

Findings:

- The results demonstrate how Django's ORM facilitates modularity and seamless schema changes while streamlining database administration by automatically generating tables from models, ma naging relationships between entities, and offering an intuitive admin interface.

Summary:

- Models for books, authors, users, and borrowing records were defined as part of the Django data base schema design exercise for a library management system.By handling relationships betwee n entities and automatically creating tables from models, Django's ORM made database interacti on easier.The development of distinct applications for users and books allowed the system to bec ome modular.Furthermore, the migration system made sure that database schema modifications went smoothly, and the Django admin interface made data maintenance simple.
  This method showed how Django's robust capabilities and user-friendliness simplify web building.

Conclusion:

- Conclusively, Django offers a fast and effective method for creating database-driven applications, such a library management system.By automatically creating tables and man aging relationships between entities, its ORM streamlines database operations, and the admin in terface provides an easy-to-use method of data management.Project organization is improved by Django programs' modular design, and database updates are made simple by the migration mechanism.All things considere d, Django's robust tools and integrated capabilities drastically cut down on complexity and devel opment time, making it the perfect option for creating scalable and maintainable web applicatio ns.