Carangue, Lizamie G.

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
```

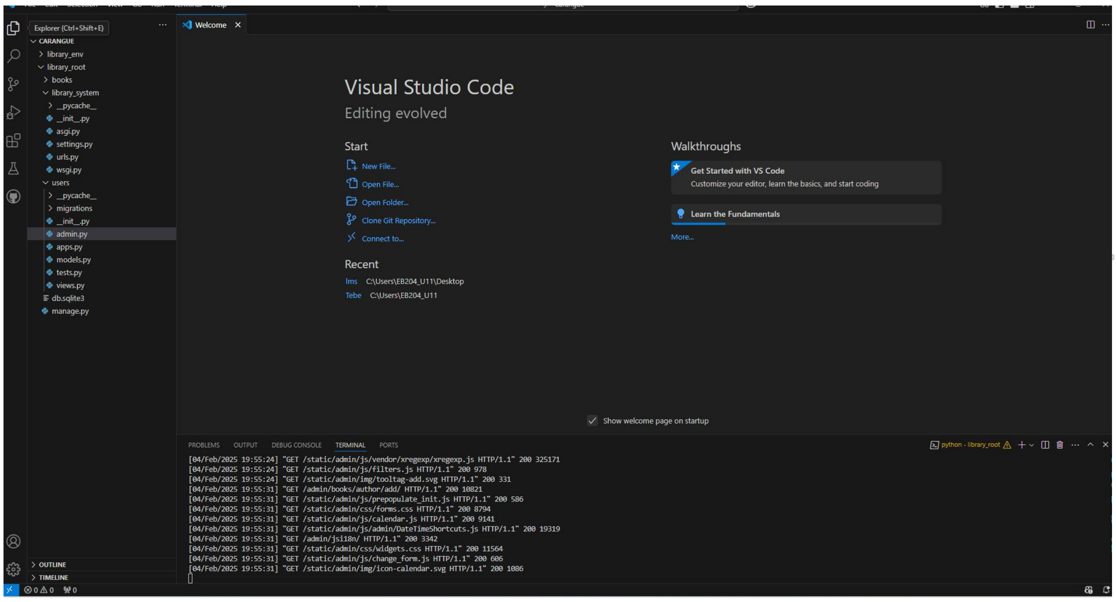
8. **Access Admin Panel**:

- 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.

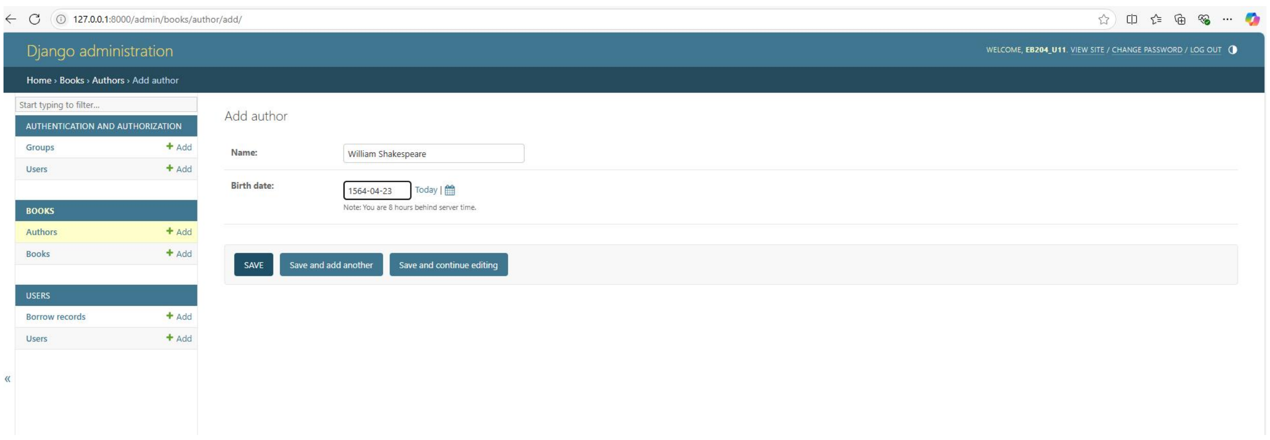
Django Program or Code: Write down the summary of the code for models that has been provided in this activity.

```
python manage.py startapp books
python manage.py startapp users
python manage.py makemigrations
python manage.py migrate
python manage.py createsuperuser
admin.site.register(Author)
admin.site.register(Book)
admin.site.register(User)
admin.site.register(BorrowRecord)
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)



```
[04/Feb/2025 19:55:24] "GET /static/admin/js/vendor/xregexp/xregexp.js HTTP/1.1" 200 325171
[04/Feb/2025 19:55:24] "GET /static/admin/js/filters.js HTTP/1.1" 200 978
[04/Feb/2025 19:55:24] "GET /static/admin/img/tooltag-add.svg HTTP/1.1" 200 331
[04/Feb/2025 19:55:31] "GET /admin/books/author/add/ HTTP/1.1" 200 10821
[04/Feb/2025 19:55:31] "GET /static/admin/js/prepopulate_init.js HTTP/1.1" 200 586
[04/Feb/2025 19:55:31] "GET /static/admin/css/forms.css HTTP/1.1" 200 8794
[04/Feb/2025 19:55:31] "GET /static/admin/js/calendar.js HTTP/1.1" 200 9141
[04/Feb/2025 19:55:31] "GET /static/admin/js/admin/DateTimeShortcuts.js HTTP/1.1" 200 19319
[04/Feb/2025 19:55:31] "GET /admin/jsi18n/ HTTP/1.1" 200 3342
[04/Feb/2025 19:55:31] "GET /static/admin/css/widgets.css HTTP/1.1" 200 11564
[04/Feb/2025 19:55:31] "GET /static/admin/js/change_form.js HTTP/1.1" 200 606
[04/Feb/2025 19:55:31] "GET /static/admin/img/icon-calendar.svg HTTP/1.1" 200 1086
```


127.0.0.1:8000/admin/books/author/add/
Django administration
WELCOME, EB204_U11. 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
SAVE
Save and add another
Save and continue editing

Add author

Name: William Shakespeare

Birth date: 1564-04-23 Today |

Note: You are 8 hours behind server time.

Follow-Up Questions:

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

A ForeignKey in Django models is used to establish a **many-to-one relationship** between two database tables. It allows one model to reference another, ensuring **data integrity and relational consistency**.

Purposes:

✅ **Establish Relationships**

- Connects models in a relational database (e.g., each book belongs to one author, but an author can have multiple books).

✅ **Enforce Data Integrity**

- Ensures that a related object must exist before referencing it (e.g., a borrowing record must reference an existing book and user).

✅ **Simplify Queries**

- Allows efficient **data retrieval and filtering** using Django's ORM. For example, book.author.name retrieves the author's name directly.

✅ **Automatic Cascade Deletions (Optional)**

- The on_delete parameter controls behavior when a referenced object is deleted.
    - models.CASCADE: Deletes related objects when the referenced object is deleted.
    - models.SET_NULL: Sets the foreign key field to NULL instead of deleting the object.
    - models.PROTECT: Prevents deletion if related objects exist.

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

✅ **1. No Need for Raw SQL**

- Developers can perform CRUD (Create, Read, Update, Delete) operations using **Python methods**, avoiding complex SQL queries.

✅ **2. Automatic Database Schema Creation**

- Defining models in Django (models.py) automatically creates the corresponding **database tables**.

✅ **3. Built-in Query Optimization**

- ORM **optimizes queries** behind the scenes to ensure efficiency.

✅ **4. Easy Relationships Handling**

- ORM automatically handles **ForeignKey, ManyToManyField, and OneToOneField** relationships.

✅ **5. Secure and Prevents SQL Injection**

- ORM **automatically escapes parameters** in queries, preventing SQL injection attacks.

✅ **6. Database Portability**

- The same ORM code works with **SQLite, PostgreSQL, MySQL, and other databases** without modification.
- Just change the DATABASES setting in settings.py.

Findings:

- The **ForeignKey** field in Django models establishes a **many-to-one** relationship, linking one model to another. For example, a **book belongs to one author, but an author can have multiple books**.
- Django’s ORM (Object-Relational Mapping) allows developers to interact with databases using **Python code instead of raw SQL**.
- ORM automatically **generates SQL queries, enforces relationships, and ensures data integrity** without requiring manual database management.
- Using Django’s ORM, developers can perform CRUD (Create, Read, Update, Delete) operations seamlessly.

Summary:

Django’s **ForeignKey** enables relational database management by linking records efficiently. Django’s **ORM simplifies database operations**, reducing the need for raw SQL and making interactions more Pythonic and readable. The ORM also ensures consistency and data integrity across related models.

Conclusion:

By leveraging **ForeignKey relationships and Django’s ORM**, database interactions become more structured, maintainable, and efficient. Django’s ORM abstracts complex SQL queries, making web development faster and more accessible. This approach **enhances productivity while ensuring data security and consistency**.