GUERRA, CHRISTINE J.

BSCpE-2A

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)

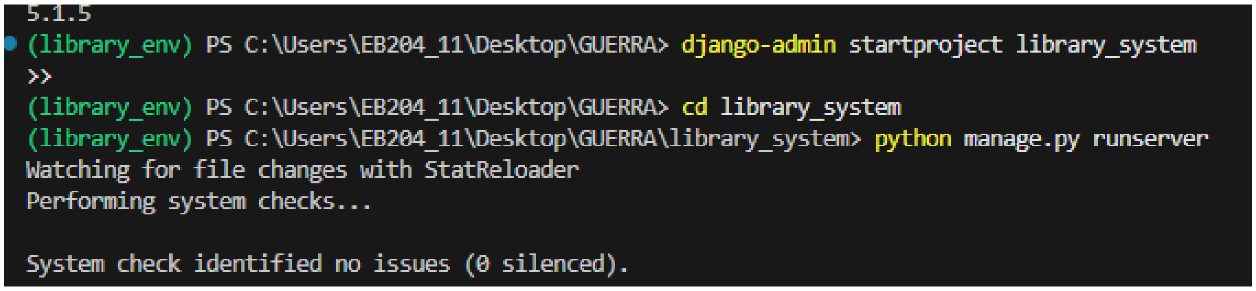
Time Frame: 2 Hours

Procedure:

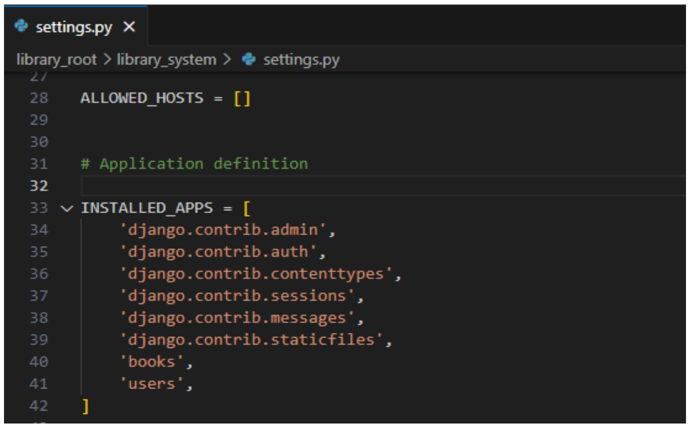
1. **Create Django Apps**:
   - In Django, an app is a module that handles a specific functionality. To keep things modular, we will create two apps: one for managing books and another for managing users.

python manage.py startapp books

python manage.py startapp users



```
5.1.5
(library_env) PS C:\Users\EB204_11\Desktop\GUERRA> django-admin startproject library_system
>>
(library_env) PS C:\Users\EB204_11\Desktop\GUERRA> cd library_system
(library_env) PS C:\Users\EB204_11\Desktop\GUERRA\library_system> python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
```



```
ALLOWED_HOSTS = []


# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'books',
    'users',
]
```

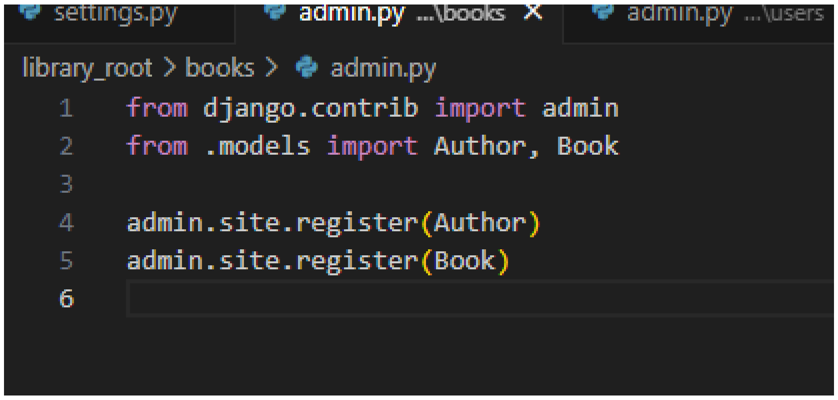
2. **Define Models for the Books App**:
   - Open the books/models.py file and define the following models:

from django.db import models

class Author(models.Model):

  name = models.CharField(max_length=100)

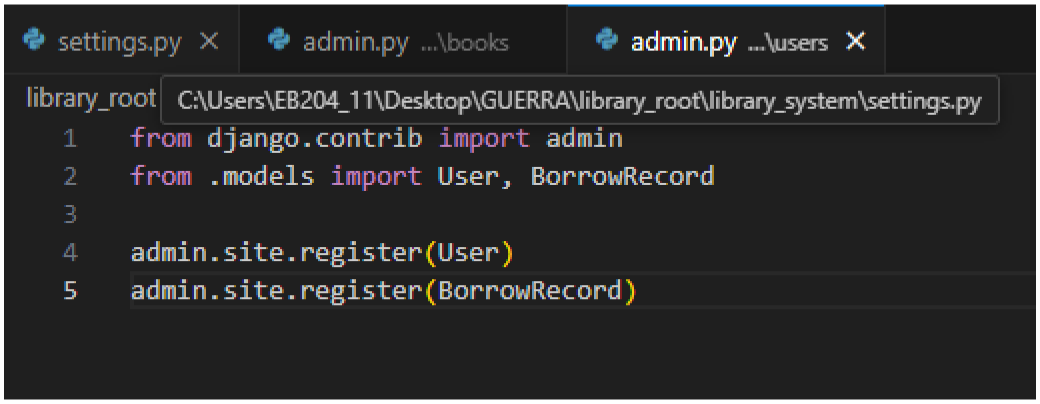
```
    birth_date = models.DateField()

    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
```



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

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

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

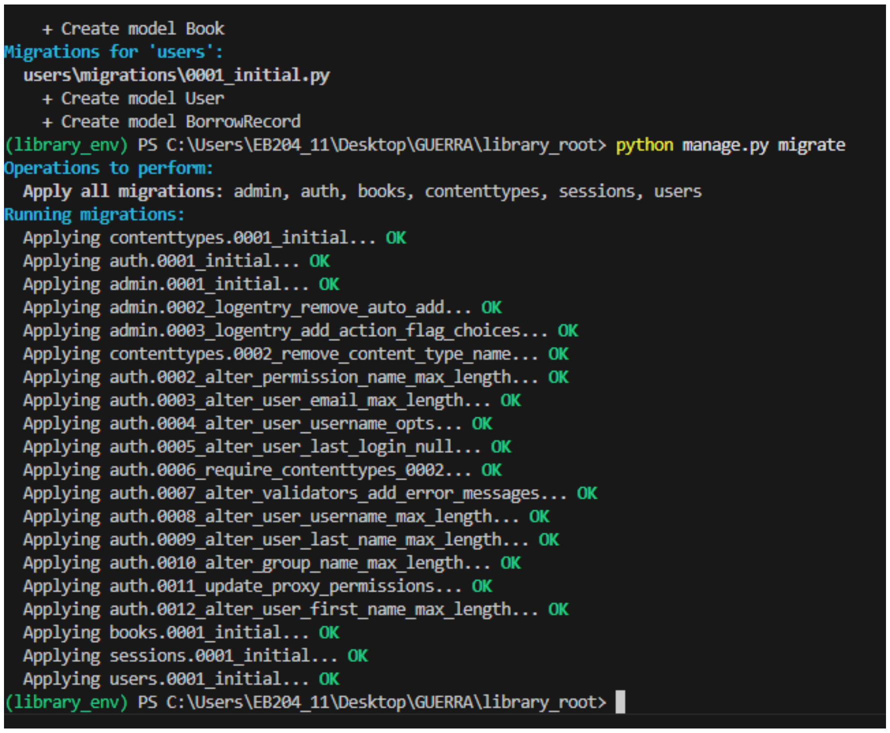
```
from django.db import models
from books.models import Book

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)
```


settings.py
admin.py ...\books
admin.py ...\users
library_root
C:\Users\EB204_11\Desktop\GUERRA\library_root\library_system\settings.py

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

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

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

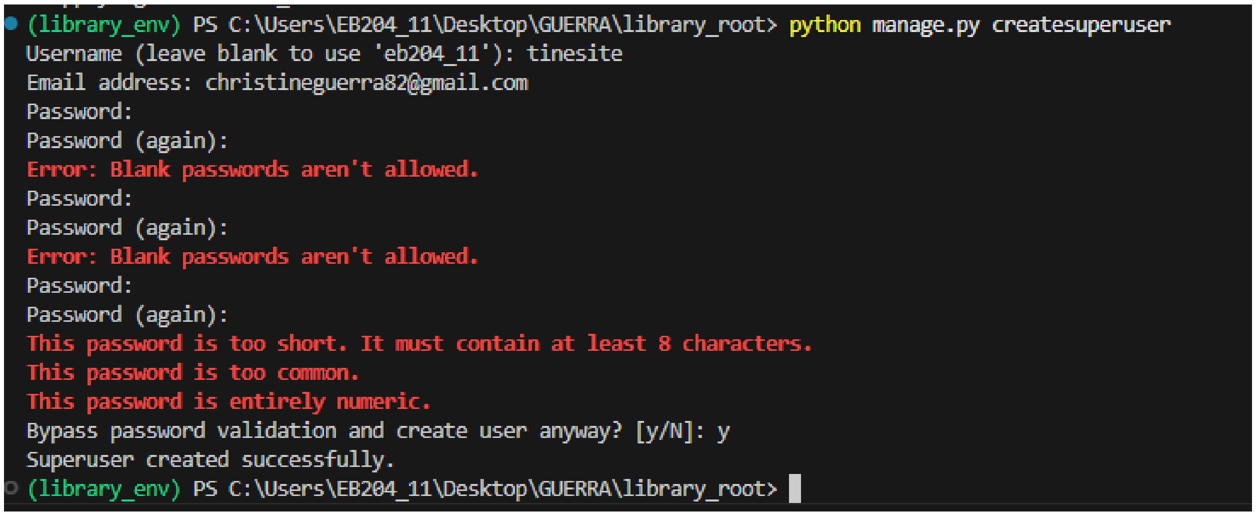
```
python manage.py makemigrations

python manage.py migrate
```



```
    + Create model Book
Migrations for 'users':
  users\migrations\0001_initial.py
    + Create model User
    + Create model BorrowRecord
(library_env) PS C:\Users\EB204_11\Desktop\GUERRA\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\EB204_11\Desktop\GUERRA\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_11\Desktop\GUERRA\library_root> python manage.py createsuperuser
Username (leave blank to use 'eb204_11'): tinesite
Email address: christineguerra82@gmail.com
Password:
Password (again):
Error: Blank passwords aren't allowed.
Password:
Password (again):
Error: Blank passwords aren't allowed.
Password:
Password (again):
This password is too short. It must contain at least 8 characters.
This password is too common.
This password is entirely numeric.
Bypass password validation and create user anyway? [y/N]: y
Superuser created successfully.
(library_env) PS C:\Users\EB204_11\Desktop\GUERRA\library_root>
```

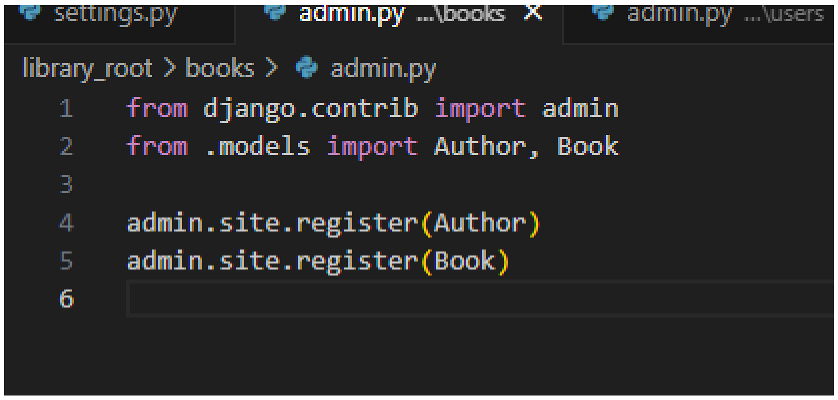
6. **Register Models in Admin Panel**:
    - In books/admin.py, register the Author and Book models:

from django.contrib import admin

from .models import Author, Book

admin.site.register(Author)

admin.site.register(Book)



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

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

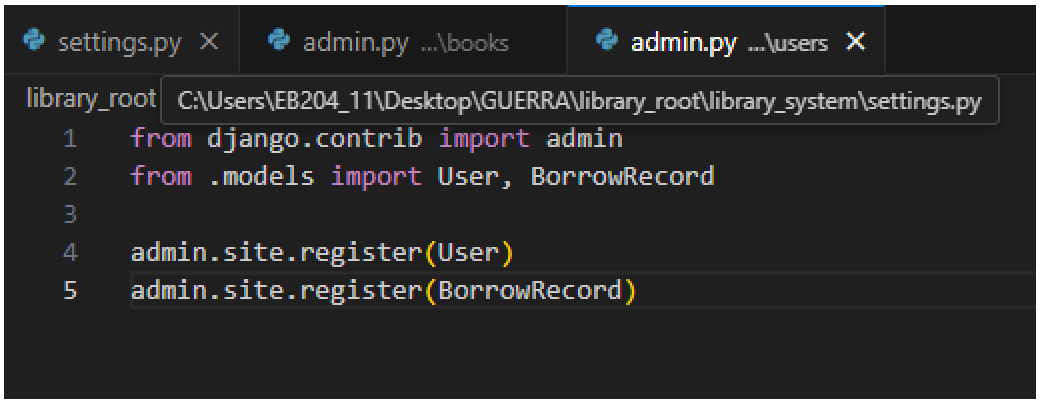
- In users/admin.py, register the User and BorrowRecord models:

from django.contrib import admin

from .models import User, BorrowRecord

admin.site.register(User)

admin.site.register(BorrowRecord)



settings.py
admin.py ...\books
admin.py ...\users
library_root
C:\Users\EB204_11\Desktop\GUERRA\library_root\library_system\settings.py

```
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:

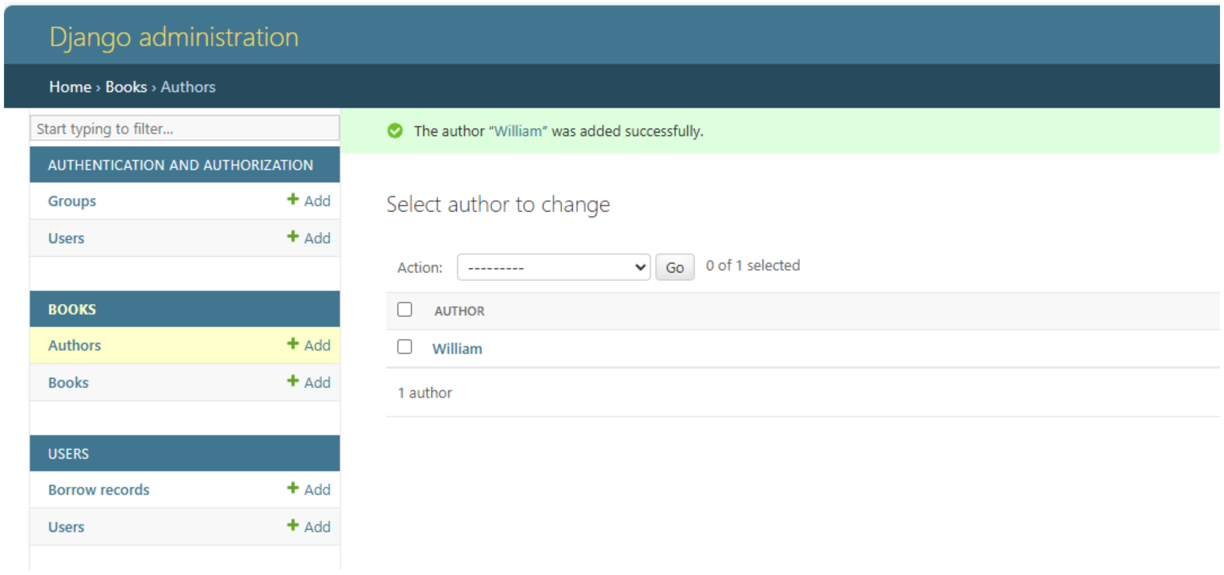
python manage.py runserver

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)


Django administration

Home › Books › Authors

The author "William" was added successfully.

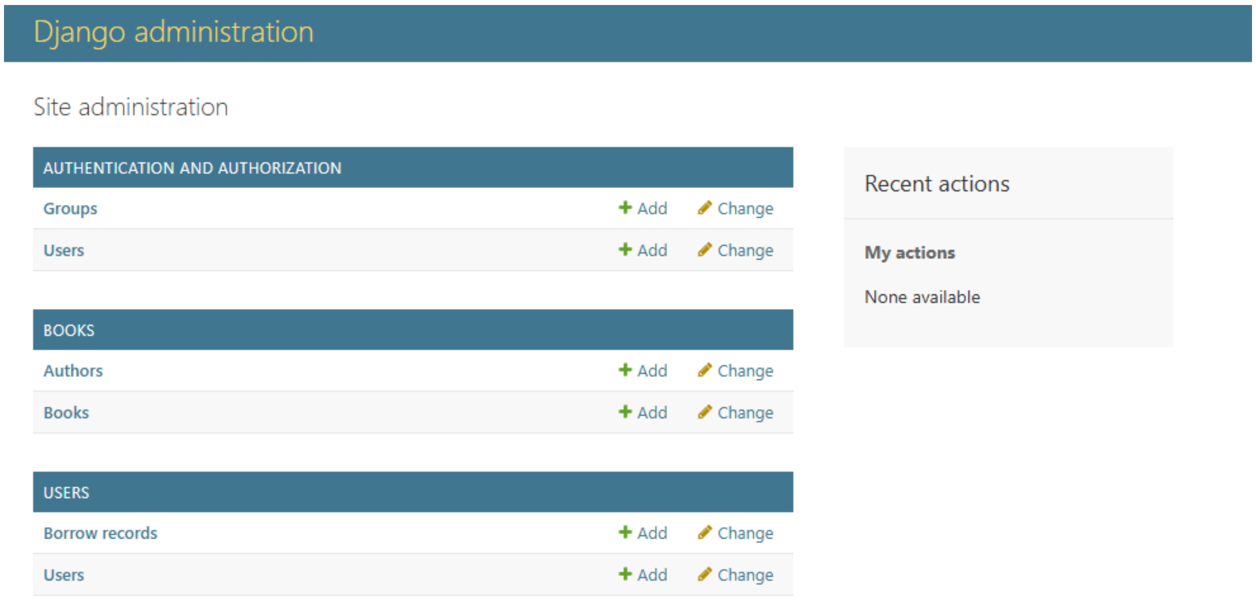
Select author to change

Action: --------- Go 0 of 1 selected

AUTHOR

William

1 author



Django administration

Site administration

AUTHENTICATION AND AUTHORIZATION		
Groups	+ Add	Change
Users	+ Add	Change

BOOKS		
Authors	+ Add	Change
Books	+ Add	Change

USERS		
Borrow records	+ Add	Change
Users	+ Add	Change

Recent actions

My actions

None available

Follow-Up Questions:

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

   Ans: A ForeignKey in Django models is used to create a relationship between two database tables. It links one model to another, allowing data from one table to reference data in another. This is useful for representing one-to-many relationships, like a blog post having multiple comments.

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

Ans: Django's ORM (Object-Relational Mapping) lets developers interact with databases using Python instead of SQL, making data management easier. It simplifies querying, updating, and deleting records while ensuring database independence across SQLite, PostgreSQL, and MySQL. This makes development faster, safer, and more efficient.

Findings:

Django's ORM simplifies database management by allowing developers to use Python instead of SQL. It makes querying, updating, and deleting records easier while supporting multiple databases. This improves development speed, security, and efficiency.

Summary:

Django’s ORM provides a simple, flexible, and database-independent way to interact with data. By eliminating the need for raw SQL queries, it enhances productivity and reduces errors, making database operations more seamless and efficient.