Name: Nic Albert B. Seraño

Course/S.Y: 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.

```
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.

The project consists of two apps: **books** and **users**.

Books App:

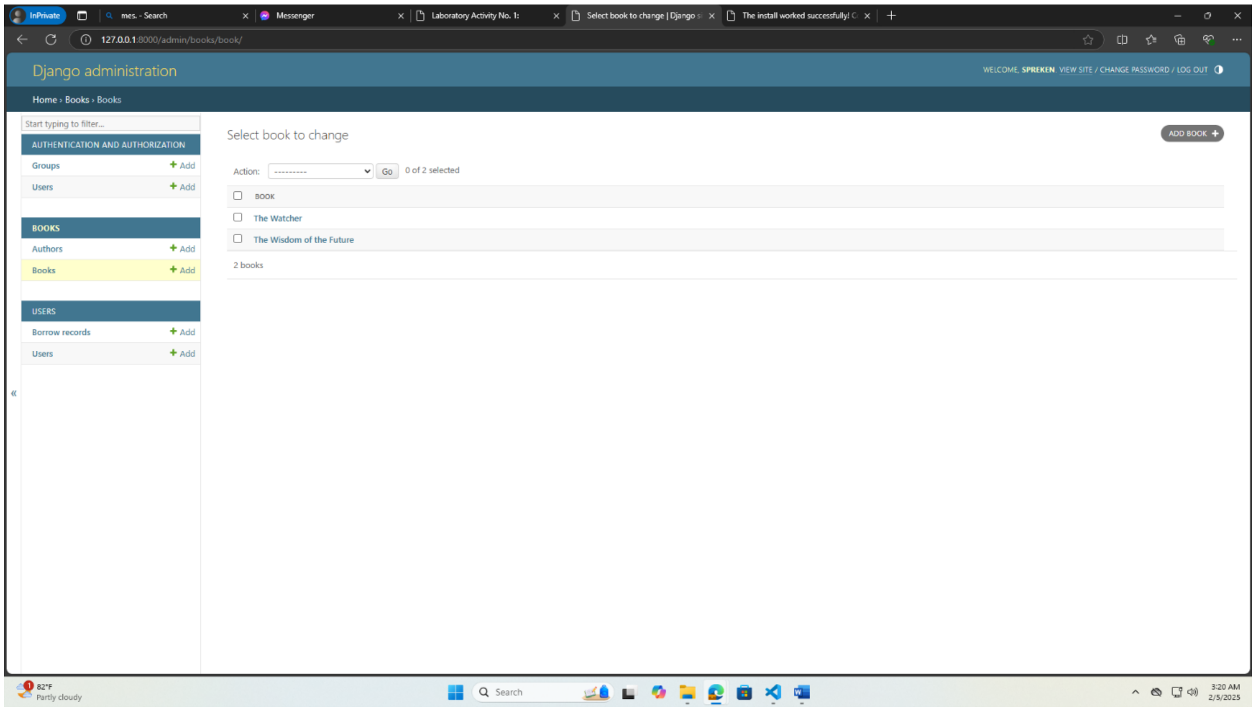
- **Author Model**: Stores the author's name and birth date.
- **Book Model**: Stores book title, author (ForeignKey to Author), ISBN, and publish date.

Users App:

- **User Model**: Stores username and email.
- **BorrowRecord Model**: Links users to books (via ForeignKeys), with borrow and return dates.

Both apps use ForeignKey relationships to connect models. The __str__ method in each model returns a readable string representation. After creating models, you run migrations, register them in the admin panel, and create a superuser to manage the system.

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)



Select book to change

0 of 2 selected

BOOK
The Watcher
The Wisdom of the Future

2 books

Github link: https://github.com/NicAlbert9/library_system

Follow-Up Questions:

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

A ForeignKey establishes a **many-to-one relationship** between two models. It links one model to another, ensuring data consistency and integrity. For example, in a library system, a Book can be linked to an Author, where one author can have many books. It also supports features like cascading deletes (e.g., deleting an author deletes their books).

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

Django's ORM allows you to interact with the database using Python objects instead of raw SQL. It automatically creates tables from models, handles relationships (like ForeignKey), and makes querying easier with methods like .filter() and .order_by(). It also helps protect against SQL injection and is database-agnostic, allowing easy switching between databases.

Findings:

- **ForeignKey** in Django models establishes a relationship between two models, linking records and ensuring data consistency.
- Django's ORM abstracts database interactions, simplifying querying, data manipulation, and managing relationships without raw SQL.

Summary:

- **ForeignKey** creates a many-to-one relationship, ensuring linked data integrity, while Django’s ORM allows database interaction using Python, making it easier and more secure.

Conclusion:

- Django’s ORM and **ForeignKey** feature significantly simplify database management, reducing the need for raw SQL, improving security, and ensuring consistency in relationships between models.