Sarvida,Jhay-r C.

BSCPE 2B1

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.

Books App Models (books/models.py)

1. **Author Model**

    Fields: name (character field, max length 100), birth_date (date field).

    Represents book authors.

    String representation returns the author's name.

2. **Book Model**

Fields: title (character field, max length 200), author (foreign key to Author with CASCADE delete behavior), isbn (character field, max length 13), publish_date (date field).

Represents books in the library.

String representation returns the book's title.

Users App Models (users/models.py)

1. **User Model**

   Fields: username (character field, max length 100), email (email field).

   Represents library users.

   String representation returns the username.

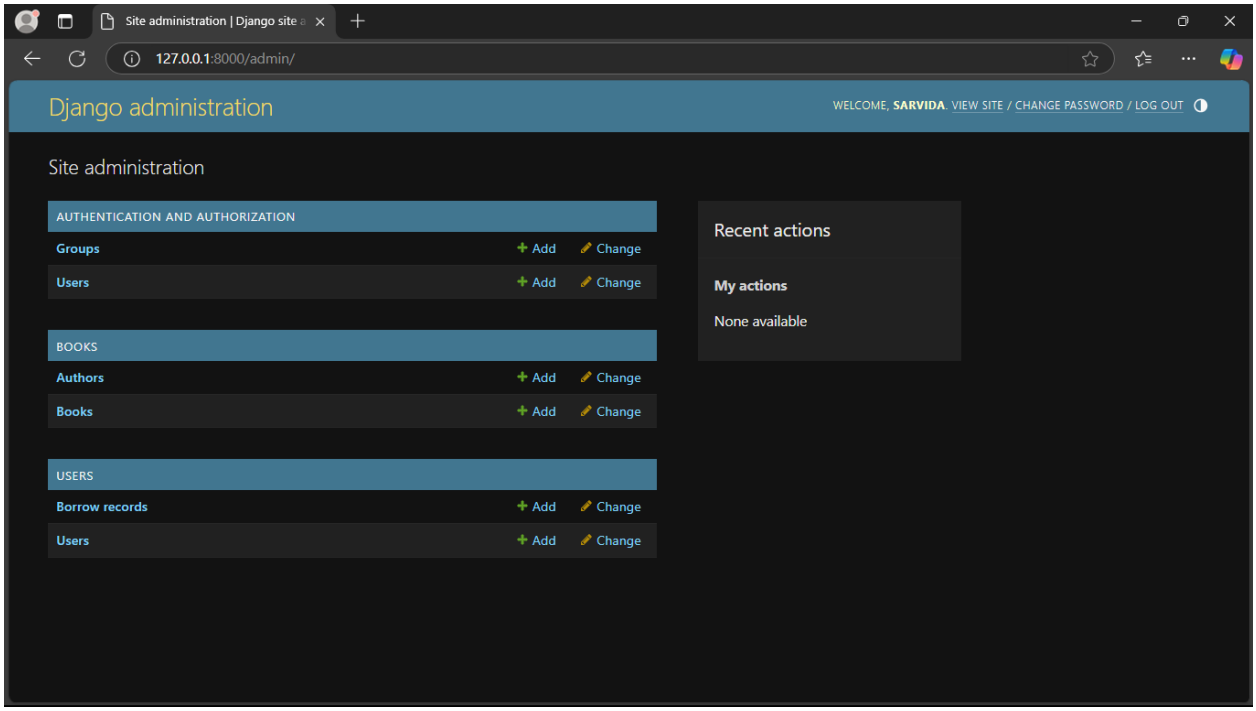
2. **BorrowRecord Model**

   Fields: user (foreign key to User with CASCADE delete behavior), book (foreign key to Book with CASCADE delete behavior), borrow_date (date field), return_date (nullable date field).

   Represents book borrowing transactions.

This schema defines a **Library Management System** where books and authors are managed in one app (books), while users and their borrowing records are handled in another (users). The relationships are set using **ForeignKey**, enforcing constraints on deletions (CASCADE), ensuring referential integrity.

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)



My Github Link: https://github.com/sarvida20/My-Project-2025/tree/99101723ff519b2e3b5623c9ba1ca663ceab3fd7/library_system

---

Follow-Up Questions:

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

Answer: A ForeignKey in Django models establishes a one-to-many relationship between two database tables. It allows one model (table) to reference another, ensuring referential integrity and enabling efficient data retrieval through relationships. This is useful for structuring related data, such as linking an author to multiple books in a library database.

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

Answer: Django's Object-Relational Mapper (ORM) abstracts raw SQL queries, allowing developers to interact with the database using Python code instead of writing SQL statements. It simplifies CRUD operations (Create, Read, Update, Delete), handles migrations, and ensures database-agnostic development, making database management easier and more efficient.

Findings:

Django's ForeignKey helps maintain relational integrity in databases.

The ORM automates query generation and data manipulation, reducing the need for raw SQL.

Using Django's ORM improves code readability, security, and maintainability.

Summary:

Django's ForeignKey field is essential for defining relationships between models, while its ORM provides a high-level abstraction for interacting with the database. These features help developers work efficiently with data without directly managing complex SQL queries.

Conclusion:

Django's ForeignKey and ORM are powerful tools for database management. The ForeignKey field enforces relationships between models, and the ORM simplifies data operations, making development faster, safer, and more scalable.