Student Management System as a CRUD (Create, Read, Update, Delete) application using Django and MySQL.

Below is the guide for you step by step, ensuring clarity for Each step.

Step 1: Setting Up Django and MySQL

Before building the application, we need to set up the environment.

1. Install Python and Django

1. Install Python:
    - Download the latest version of Python from python.org.
    - Follow the installation steps, ensuring you add Python to your system's PATH.
    - Check if Python is installed by running:

python --version

2. Install Django:
    - Open a terminal/command prompt and type:

pip install django

    - Verify Django installation:

django-admin --version

2. Install and Configure MySQL

1. Install MySQL:
    - Download MySQL from mysql.com and install it.
2. Install the Python MySQL client:
    - Use the command:

pip install mysqlclient

    - This allows Django to connect to MySQL.

3. Create a Django Project

1. Start a new Django project:

django-admin startproject student_management

cd student_management

2. Run the development server to test:

python manage.py runserver

- Open a browser and navigate to http://127.0.0.1:8000/. You should see the Django welcome page.

---

4. Configure MySQL as the Database

1. Open the settings.py file in the student_management folder.
2. Update the DATABASES configuration:

```
DATABASES = {
   'default': {
      'ENGINE': 'django.db.backends.mysql',
      'NAME': 'student_db',
      'USER': 'your_mysql_username',
      'PASSWORD': 'your_mysql_password',
      'HOST': 'localhost',
      'PORT': '3306',
   }
}
```

3. Create the database in MySQL:

   Open Mysql Workbench

   - Create a database:

     CREATE DATABASE student_db;

4. Apply Django migrations:

python manage.py makemigrations

python manage.py migrate

Check Outputs for this Step:

- **python manage.py runserver:**

Watching for file changes with StatReloader

Performing system checks...

System check identified no issues (0 silenced).

Django version 4.x.x, using settings 'student_management.settings'

Starting development server at http://127.0.0.1:8000/

Quit the server with CONTROL-C.

- Opening the browser to http://127.0.0.1:8000/ should show the Django welcome page.

Step 2: Creating the Student Model

Now that the Django project is set up and connected to MySQL, we'll define a **Student Model** to represent student records in the database. A **Model** in Django is a Python class that defines the structure of database tables.

---

1. Create a Django App

Django organizes functionality into modular **apps**. Let's create an app named students.

1. Run the following command in the terminal:

python manage.py startapp students

2. Add the students app to the INSTALLED_APPS section in settings.py:

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'students',  # Add this line
]
```

---

2. Define the Student Model

Open the models.py file in the students app folder and define the Student model:

```
from django.db import models
```

```
class Student(models.Model):
    first_name = models.CharField(max_length=50)
    last_name = models.CharField(max_length=50)
    email = models.EmailField(unique=True)
    date_of_birth = models.DateField()
    grade = models.CharField(max_length=10)
    address = models.TextField()

    def __str__(self):
        return f"{self.first_name} {self.last_name}"
```

---

3. Explanation of the Fields

- **first_name and last_name**: Store the student's first and last names. We use CharField with a maximum length of 50 characters.
- **email**: A unique email address for the student using EmailField.
- **date_of_birth**: The student's date of birth using DateField.
- **grade**: The student's current grade or class, stored as a short text.
- **address**: A detailed address using TextField, which is suitable for long text.
- **__str__ method**: Returns a string representation of the student, making it easier to identify in the Django admin interface.

---

4. Apply Migrations

1. Create migration files for the Student model:

python manage.py makemigrations students

   - Example output:

```
Migrations for 'students':
  students/migrations/0001_initial.py - Create model Student
```

2. Apply the migrations to the database:

python manage.py migrate

   - Example output:

Applying students.0001_initial... OK

---

5. Test the Model in the Django Shell

1. Open the Django shell:

```
python manage.py shell
```

2. Create a new student record:

```
from students.models import Student
student = Student.objects.create(
   first_name="John",
   last_name="Doe",
   email="johndoe@example.com",
   date_of_birth="2005-08-15",
   grade="10",
   address="123 Elm Street, Springfield"
)
print(student)
```

- Expected output:

John Doe

3. Query all student records:

```
students = Student.objects.all()
print(students)
```

What We Achieved in This Step

1. Created the students app.
2. Defined the Student model with fields for essential student data.
3. Migrated the model to create a students_student table in MySQL.
4. Verified the model by creating and querying a record in the Django shell.

Step 3: Adding the Student Model to the Admin Panel

In this step, we'll configure the **Django Admin Panel** to manage student records easily through a web interface. This will allow us to create, update, and delete student records visually.

---

1. Enable the Admin Panel

Django includes a built-in admin interface for managing models. Let's register the Student model so it appears in the admin panel.

1. Open admin.py in the students app folder.
2. Register the Student model:

```
from django.contrib import admin

from .models import Student


@admin.register(Student)

class StudentAdmin(admin.ModelAdmin):

  list_display = ('first_name', 'last_name', 'email', 'grade')

  search_fields = ('first_name', 'last_name', 'email')

  list_filter = ('grade', 'date_of_birth')
```

---

2. Explanation of Admin Configuration

- **@admin.register(Student)**: Registers the Student model with the admin panel.
- **list_display**: Specifies which fields to show in the list view of students.
- **search_fields**: Enables a search box to filter students by first name, last name, or email.
- **list_filter**: Adds filters on the right-hand side to narrow results by grade or date of birth.

---

3. Create a Superuser

To access the admin panel, you need a superuser account.

1. Run the following command:

```
python manage.py createsuperuser
```

2. Enter the details when prompted:

- **Username**: admin
- **Email**: admin@example.com
- **Password**: Choose a strong password and confirm.

---

4. Access the Admin Panel

1. Start the development server if it's not running:

python manage.py runserver

2. Open a browser and navigate to:

http://127.0.0.1:8000/admin/

3. Log in using the superuser credentials you created.

---

5. Test the Admin Panel

1. After logging in, you'll see the Student model listed.
2. Click **Students** to view, add, or edit records.
3. Add a new student:
   - Click **Add Student**.
   - Fill in the fields (e.g., First Name: Alice, Last Name: Smith, etc.).
   - Click **Save**.
4. View the student list:
   - You'll see the list of students with the fields specified in list_display.

Example Outputs

- **Admin Panel Login Screen**: A form to enter your username and password.
- **Student List in the Admin Panel**:

| First Name | Last Name | Email | Grade |

|---|---|---|---|

| Alice | Smith | alice@example.com | 10 |

| John | Doe | johndoe@example.com | 10 |

What We Achieved in This Step

1. Registered the Student model in the Django Admin Panel.
2. Customized the admin interface to display key fields and add filtering and search capabilities.
3. Verified the functionality by adding and viewing student records in the admin panel.

Step 4: Setting Up Views and URLs for CRUD Operations (Manual Way)

Instead of Django's generic class-based views, we'll write function-based views for each CRUD operation. This way, we have full control over the logic and can learn the underlying mechanisms.

---

1. Create Function-Based Views for CRUD

Open the views.py file in the students app and define the following views:

---

a. List Students (Read Operation)

This view retrieves all students from the database and passes them to a template.

```
from django.shortcuts import render, get_object_or_404, redirect

from .models import Student


def student_list(request):
    # Query all student records
    students = Student.objects.all()
    # Render the list template with the student data
    return render(request, 'students/student_list.html', {'students': students})
```

---

b. Add a Student (Create Operation)

This view handles both GET (show form) and POST (process form data) requests.

```
def student_add(request):
    if request.method == 'POST':
        # Extract data from the form
```

```
        first_name = request.POST.get('first_name')
        last_name = request.POST.get('last_name')
        email = request.POST.get('email')
        date_of_birth = request.POST.get('date_of_birth')
        grade = request.POST.get('grade')
        address = request.POST.get('address')

        # Create a new student
        Student.objects.create(
            first_name=first_name,
            last_name=last_name,
            email=email,
            date_of_birth=date_of_birth,
            grade=grade,
            address=address
        )
        # Redirect to the student list after saving
        return redirect('student-list')

    # If GET request, render the form
    return render(request, 'students/student_form.html')
```

---

c. Edit a Student (Update Operation)

This view loads a student record for editing and updates it on form submission.

```
def student_edit(request, pk):
    # Fetch the student by primary key (ID)
    student = get_object_or_404(Student, pk=pk)

    if request.method == 'POST':
```

```
        # Update the student details
        student.first_name = request.POST.get('first_name')
        student.last_name = request.POST.get('last_name')
        student.email = request.POST.get('email')
        student.date_of_birth = request.POST.get('date_of_birth')
        student.grade = request.POST.get('grade')
        student.address = request.POST.get('address')
        student.save()  # Save changes to the database
        return redirect('student-list')

    # Render the form with existing data
    return render(request, 'students/student_form.html', {'student': student})
```

---

d. Delete a Student (Delete Operation)

This view confirms the deletion and removes the record on confirmation.

```
def student_delete(request, pk):
    # Fetch the student by primary key (ID)
    student = get_object_or_404(Student, pk=pk)

    if request.method == 'POST':
        student.delete()  # Delete the student record
        return redirect('student-list')

    # Render the delete confirmation page
    return render(request, 'students/student_confirm_delete.html', {'student': student})
```

---

2. Configure URLs for CRUD

Update the urls.py file in the students app to map these views:

```
from django.urls import path
```

```
from . import views

urlpatterns = [
    path('', views.student_list, name='student-list'),          # List students
    path('add/', views.student_add, name='student-add'),        # Add a student
    path('<int:pk>/edit/', views.student_edit, name='student-edit'),  # Edit a student
    path('<int:pk>/delete/', views.student_delete, name='student-delete'),  # Delete a student
]
```

---

3. Create or Update Templates

We'll reuse the same templates but modify them slightly to work with the manual views.

---

a. Template for Listing Students (student_list.html)

Displays all students and provides links to add, edit, and delete.

```
<!DOCTYPE html>
<html>
<head>
    <title>Student List</title>
</head>
<body>
    <h1>Student List</h1>
    <a href="{% url 'student-add' %}">Add New Student</a>
    <table border="1">
        <thead>
            <tr>
                <th>First Name</th>
                <th>Last Name</th>
                <th>Email</th>
                <th>Grade</th>
```

```
            <th>Actions</th>
          </tr>
        </thead>
        <tbody>
          {% for student in students %}
          <tr>
            <td>{{ student.first_name }}</td>
            <td>{{ student.last_name }}</td>
            <td>{{ student.email }}</td>
            <td>{{ student.grade }}</td>
            <td>
              <a href="{% url 'student-edit' student.pk %}">Edit</a> |
              <a href="{% url 'student-delete' student.pk %}">Delete</a>
            </td>
          </tr>
          {% endfor %}
        </tbody>
    </table>
</body>
</html>
```

---

b. Template for the Form (student_form.html)

Displays the form for adding or editing students.

```
<!DOCTYPE html>
<html>
<head>
    <title>Student Form</title>
</head>
<body>
```

```
    <h1>{% if student %}Edit{% else %}Add{% endif %} Student</h1>
    <form method="post">
        {% csrf_token %}
        <label for="first_name">First Name:</label>
        <input type="text" name="first_name" value="{{ student.first_name|default:'' }}"><br>

        <label for="last_name">Last Name:</label>
        <input type="text" name="last_name" value="{{ student.last_name|default:'' }}"><br>

        <label for="email">Email:</label>
        <input type="email" name="email" value="{{ student.email|default:'' }}"><br>

        <label for="date_of_birth">Date of Birth:</label>
        <input type="date" name="date_of_birth" value="{{ student.date_of_birth|default:''
}}"><br>

        <label for="grade">Grade:</label>
        <input type="text" name="grade" value="{{ student.grade|default:'' }}"><br>

        <label for="address">Address:</label>
        <textarea name="address">{{ student.address|default:'' }}</textarea><br>

        <button type="submit">Save</button>
    </form>
    <a href="{% url 'student-list' %}">Back to List</a>
</body>
</html>
```

---

c. Template for Delete Confirmation (student_confirm_delete.html)

Asks for confirmation before deleting a student.

```
<!DOCTYPE html>
<html>
<head>
  <title>Delete Student</title>
</head>
<body>
  <h1>Are you sure you want to delete "{{ student }}"?</h1>
  <form method="post">
    {% csrf_token %}
    <button type="submit">Yes, Delete</button>
  </form>
  <a href="{% url 'student-list' %}">Cancel</a>
</body>
</html>
```

---

4. Test the Application

1. Start the server:

python manage.py runserver

2. Access the application at:

http://127.0.0.1:8000/students/

3. Test each operation:
    - **Add** a student.
    - **View** the list.
    - **Edit** an existing record.
    - **Delete** a record.

What We Achieved

- Created CRUD functionality with detailed logic in the views.
- Gave students a hands-on understanding of how data flows in Django.

Step 5: Adding Styling Using Bootstrap

---

1. Integrate Bootstrap in Your Project

To use Bootstrap, we'll include the Bootstrap CSS and JS files from the official CDN.

Modify your base HTML structure to include these links.

Create a Base Template

First, create a base template base.html in the templates/students directory. This will serve as a layout for all other templates.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>{% block title %}Student Management System{% endblock %}</title>
    <!-- Bootstrap CSS -->
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-
alpha3/dist/css/bootstrap.min.css" rel="stylesheet">
</head>
<body>
    <nav class="navbar navbar-expand-lg navbar-dark bg-dark">
        <div class="container-fluid">
            <a class="navbar-brand" href="{% url 'student-list' %}">Student Management</a>
        </div>
    </nav>
    <div class="container mt-4">
        {% block content %}{% endblock %}
    </div>
```

```
    <!-- Bootstrap Bundle JS -->
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha3/dist/js/bootstrap.bundle.min.js"></script>
</body>
</html>
```

---

2. Update Templates to Use Bootstrap

Now we'll modify the existing templates to use the base.html and include Bootstrap classes for styling.

---

a. Student List Template (student_list.html)

```
{% extends 'students/base.html' %}

{% block title %}Student List{% endblock %}

{% block content %}
<h1 class="mb-4">Student List</h1>
<a href="{% url 'student-add' %}" class="btn btn-primary mb-3">Add New Student</a>
<table class="table table-striped table-bordered">
    <thead class="table-dark">
        <tr>
            <th>First Name</th>
            <th>Last Name</th>
            <th>Email</th>
            <th>Grade</th>
            <th>Actions</th>
        </tr>
    </thead>
    <tbody>
        {% for student in students %}
```

```
    <tr>
      <td>{{ student.first_name }}</td>
      <td>{{ student.last_name }}</td>
      <td>{{ student.email }}</td>
      <td>{{ student.grade }}</td>
      <td>
        <a href="{% url 'student-edit' student.pk %}" class="btn btn-warning btn-sm">Edit</a>
        <a href="{% url 'student-delete' student.pk %}" class="btn btn-danger btn-sm">Delete</a>
      </td>
    </tr>
    {% endfor %}
  </tbody>
</table>
{% endblock %}
```

---

b. Student Form Template (student_form.html)

```
{% extends 'students/base.html' %}

{% block title %}{% if student %}Edit{% else %}Add{% endif %} Student{% endblock %}

{% block content %}
<h1>{% if student %}Edit{% else %}Add{% endif %} Student</h1>
<form method="post" class="row g-3">
  {% csrf_token %}
  <div class="col-md-6">
    <label for="first_name" class="form-label">First Name</label>
    <input type="text" name="first_name" class="form-control" value="{{ student.first_name|default:'' }}">
```

```
    </div>
    <div class="col-md-6">
        <label for="last_name" class="form-label">Last Name</label>
        <input type="text" name="last_name" class="form-control" value="{{
student.last_name|default:'' }}">
    </div>
    <div class="col-md-6">
        <label for="email" class="form-label">Email</label>
        <input type="email" name="email" class="form-control" value="{{
student.email|default:'' }}">
    </div>
    <div class="col-md-6">
        <label for="date_of_birth" class="form-label">Date of Birth</label>
        <input type="date" name="date_of_birth" class="form-control" value="{{
student.date_of_birth|default:'' }}">
    </div>
    <div class="col-md-6">
        <label for="grade" class="form-label">Grade</label>
        <input type="text" name="grade" class="form-control" value="{{
student.grade|default:'' }}">
    </div>
    <div class="col-12">
        <label for="address" class="form-label">Address</label>
        <textarea name="address" class="form-control">{{ student.address|default:''
}}</textarea>
    </div>
    <div class="col-12">
        <button type="submit" class="btn btn-success">Save</button>
        <a href="{% url 'student-list' %}" class="btn btn-secondary">Cancel</a>
    </div>
</form>
```

{% endblock %}

---

c. Delete Confirmation Template (student_confirm_delete.html)

{% extends 'students/base.html' %}

{% block title %}Delete Student{% endblock %}

{% block content %}

<h1>Confirm Deletion</h1>

Are you sure you want to delete {{ student.first_name }} {{ student.last_name }}?

<form method="post">

{% csrf_token %}

<button type="submit" class="btn btn-danger">Yes, Delete</button>

Cancel

</form>

{% endblock %}

---

3. Test the Application

1. Restart the server:

python manage.py runserver

2. Visit the application at:

http://127.0.0.1:8000/students/

3. Notice the enhanced look and feel:
    - Buttons, tables, and forms now look polished and responsive.
    - Navigation bar and layout make the UI professional.