College Enterprise Resource Planner

Functional Web Application

—

Prabhav Sunil Patil
Roll Number: 20MA20042

DBMS Lab Report

College-ERP Database

Abstract

This is a College Enterprise Resource Planner database, which is developed by making use of the Python/Django Framework for building a fully functional web application.

Features of the Database:

A. Admin Users Can :

1. Manage Staff (Add, Update and Delete)
2. Manage Students (Add, Update and Delete)
3. Manage Course (Add, Update and Delete)
4. Manage Enrollments (Add, Update and Delete)
5. Manage Prerequisites for a Course (Add, Update and Delete)
6. Manage Departments (Addm Update and Delete)

B. Staff/Teachers Can :

1. View the Courses that they are currently taking
2. View the Enrollments (Student Name, Enrollment Date and Grade) in their taken courses

C. Students Can :

1. View the Courses that they are currently enrolled in
2. View the Courses that they can take, i.e., the courses for which they have completed the prerequisite for.

Database Schema

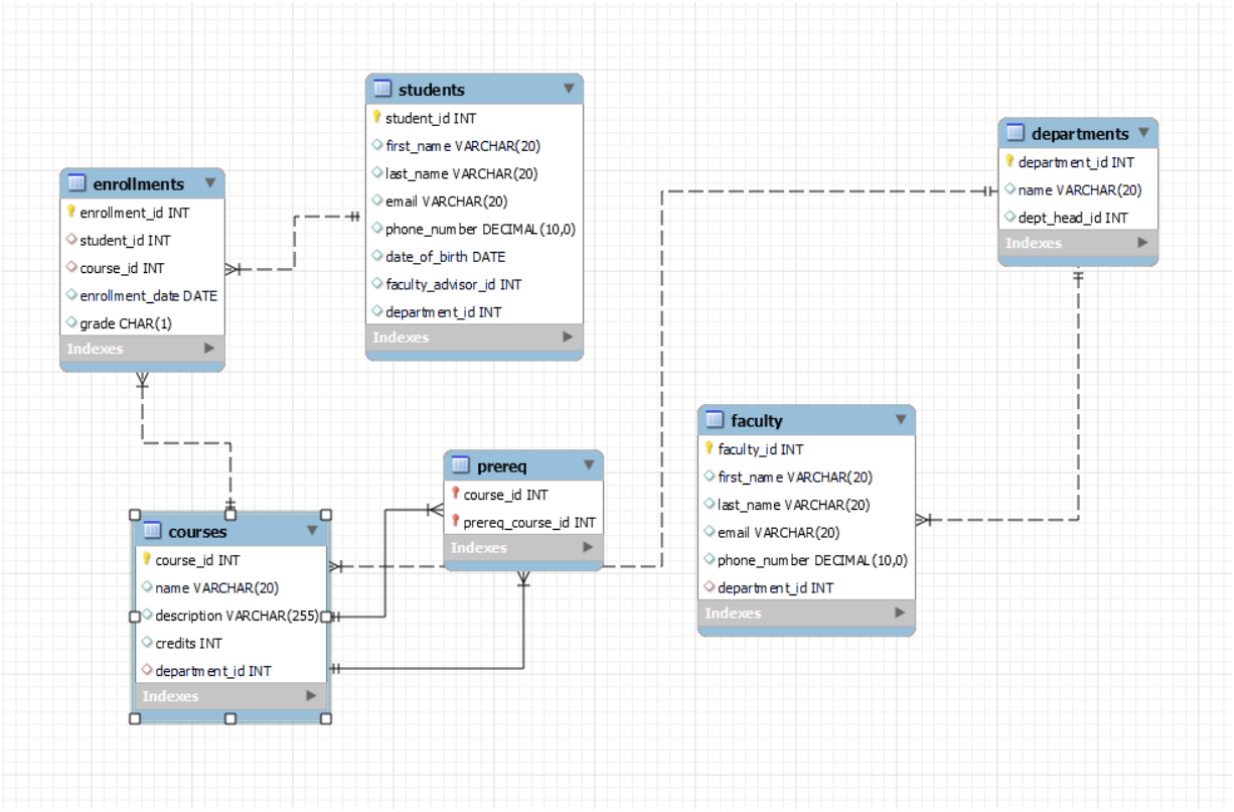
We create the tables: **Department, Faculty, Student, Course, Enrollment, Prereq** which are used to store the data efficiently and conveniently.

The database is created in **models.py** file which then integrates with the Django framework so as to create the whole structure of the database.

```python
class faculty(models.Model):
    faculty_id = models.IntegerField(primary_key = True)
    name = models.CharField(max_length = 100)
    email = models.EmailField()
    phone_number = models.CharField(max_length = 10)
    department_id = models.ForeignKey(department, on_delete = models.CASCADE)
    password = models.CharField(max_length = 100, default = 'password')

    def __str__(self):
        return str(self.faculty_id)

class student(models.Model):
    roll_number = models.CharField(max_length = 9, primary_key = True)
    name = models.CharField(max_length = 100)
    email = models.EmailField()
    date_of_birth = models.DateField()
    facad_id = models.ForeignKey(faculty, on_delete = models.CASCADE)
    department_id = models.ForeignKey(department, on_delete = models.CASCADE)
    password = models.CharField(max_length = 100, default = 'password')

    def __str__(self):
        return self.roll_number

class course(models.Model):
    course_id = models.IntegerField(primary_key = True)
    name = models.CharField(max_length = 100)
    credits = models.IntegerField()
    faculty_id = models.ForeignKey(faculty, on_delete = models.CASCADE)
    department_id = models.ForeignKey(department, on_delete = models.CASCADE)

    def __str__(self):
        return self.name

class enrollment(models.Model):
    enrollment_id = models.IntegerField(primary_key = True)
    roll_number = models.ForeignKey(student, on_delete = models.CASCADE)
    course_id = models.ForeignKey(course, on_delete = models.CASCADE)
    enrollment_date = models.DateField()
    grade = models.IntegerField()

    def __str__(self):
        return str(self.enrollment_id)

class prereq(models.Model):
    c_id = models.ForeignKey(course, on_delete = models.CASCADE, related_name='%(class)s_requests_created')
    p_id = models.ForeignKey(course, on_delete = models.CASCADE, related_name='%(class)s_requests_assigned')

    def __str__(self):
        return str(self.c_id) + " " + str(self.p_id)
```

Enrollments store the information about Roll number of the student, the corresponding course which he took, and the grade as well as the date corresponding to the course enrolled. Prerequisite stores the information about the pre-requisite course which a student must take to enroll in a given course.

E-R Diagram for the Database:



Making Entries into the Database

We enter the data entries into our database using the pre-built django functions, where we declare a variable which we assign to the entries which we want to put into our database and then save the variable using the save() function.

We use a python library called **faker** library which generates fake names when called to add the entries into the database. This was done to ensure that we have large enough data already added into our database. We provide the following snippet as an example:

```
import Faker from faker()
import random
fake = Faker()

s1 = fake.first_name()
s2 = fake.last_name()

for x in range(10,99):
    a1 = students.objects.get_or_create(roll_number = '20MA200'+str(i), name = s1+" "+s2, email=s1+'.'+s2+'@gmail.com', \
    phone_number = 9823562237, department_id = 6, password = 'password')
    a1.save()
```

Integrating the Backend with the Frontend

I. Views.py

Django views are Python functions that take http requests and return http response, like HTML documents.

We then define various view functions which take in the http request and corresponding to it, return the HTML documents giving it a context by generating the set of values using the queries in the view functions.

There are many view functions written such as home (for the homepage), student_enrollable_courses, faculty_by_department, enrolled_courses etc. Some snippets for the view functions is shown as follows:

```python
def home(request):
    return render(request, "home.html")

def faculty_course(request, faculty_id):
    courses = course.objects.filter(faculty_id=faculty_id)
    context = {
        'courses': courses,
        'faculty_id': faculty_id
    }
    return render(request, 'faculty_courses.html', context)

def enrollmentview(request, course_id):
        enrollments = enrollment.objects.filter(course_id=course_id)
        students = []
        for enroll in enrollments:
            stud = student.objects.get(roll_number=enroll.roll_number)
            students.append({
                'name': stud.name,
                'roll_number': stud.roll_number,
                'enrollment_date': enroll.enrollment_date,
                'grade': enroll.grade,
            })
        context = {
            'students': students,
            'course_id': course_id
        }
        return render(request, 'course_enrollment.html', context)

def student_enrollable_courses(request, roll_number):
    # Get all courses from the Course table
    all_courses = course.objects.all()

    # Get courses the student has already enrolled in
    enrolled_courses = enrollment.objects.filter(roll_number=roll_number).values_list('course_id', flat=True)

    # Get courses that have prerequisites and filter them based on the student's previous enrollments
    courses_with_prereqs = prereq.objects.filter().values_list('c_id', flat=True)
    eligible_courses = course.objects.exclude(course_id__in=enrolled_courses).exclude(course_id__in=courses_with_prereqs)

    context = {'roll_number': roll_number, 'courses': eligible_courses}

    return render(request, 'student_courses.html', context)
```

II. Urls.py

A request in Django first comes to urls.py and then goes to the matching function in views.py. Python functions in views.py take the web request from urls.py and give the web response to templates.

Hence, corresponding to our views function, we define a path to the template which is being returned through our view functions, and a name to the url. We can the path defined as follows:

```
urlpatterns = [
    path('admin/', admin.site.urls),
    path('erp/faculty/<int:faculty_id>/courses/', views.faculty_course, name='faculty_course'),
    path('erp/student/<str:roll_number>/available/', views.student_enrollable_courses, name='student_enrollable_courses'),
    path('erp/course/<int:course_id>/enrollment/', views.enrollmentview, name='enrollmentview'),
    path('erp/department/<int:department_id>/faculty/', views.faculty_by_department, name='faculty_by_department'),
    path('erp/department/names/', views.department_names, name='department_names'),
    path('erp/student/<str:roll_number>/enrolled/', views.enrolled_courses, name='enrolled_courses'),
    path('erp/faculty/login/', views.faculty_login, name='faculty_login'),
    path('erp/student/login/', views.student_login, name='student_login'),
    path('login/', views.login_view, name='login'),
    path('student/logout/', views.student_logout, name='student_logout'),
    path('faculty/logout/', views.faculty_logout, name='faculty_logout'),
    path('student/<str:student_id>/dashboard/', views.student_dashboard, name='student_dashboard'),
    path('faculty/<int:faculty_id>/dashboard/', views.faculty_dashboard, name='faculty_dashboard'),
    path('', views.home, name='home'),
]
```

III. Frontend (.html files)

Finally, we create various .html files corresponding to each webpage, which correspond to the templates.
A template contains the static parts of the desired HTML output as well as some special syntax describing how dynamic content will be inserted.
Hence, corresponding to our urls and views, we generate various templates which then determine how the sets of fields generated by related queries would be visualized in the frontend, i.e., on the webpage.

For every template, there is a creation of a 'base.html' file which gives the base output for all the templates which are being rendered in this project. The snippet for this is as follows:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>{% block title %}My Site{% endblock %}</title>
</head>
<body>
    <header>
        <nav>
            <ul>
                <a href= "{% url 'home' %}">Home</a>
            </ul>
        </nav>
    </header>
    <main>
        {% block content %}
        {% endblock %}
    </main>
    <footer>
        <p>&copy; 2023 My Site</p>
    </footer>
</body>
</html>
```

Final Results:

The final web pages for the prototype for college enterprise resource planning are shown here.