DBMS Project Tutorial

By Tzu-Heng Huang

Content

1. An Overview of Project Planning
2. Start your project
3. Three Layers in Web Application
4. Application Layer
5. Front-End Layer
6. HTTP Methods
7. File Structure
8. Example

1. An Overview of Project Planning

(1) Requirement analysis,

(2) ER Model Design

(3) Relational Schema Design.

(4) Setup your web framework and DBMS.

(5) Create your DB.

(6) Launch your web framework and connect to your DB.

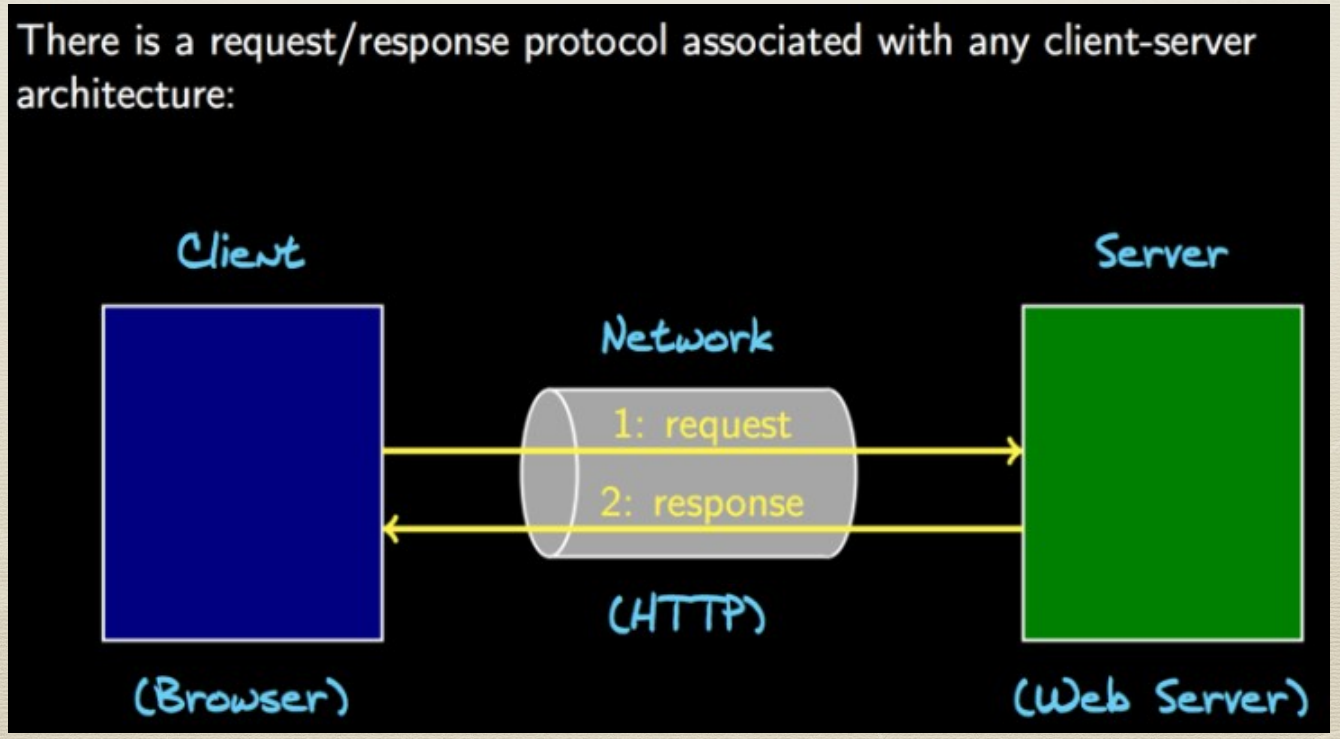
(7) Design web interface with frond-end tools.

(8) Implement CRUD functions by communicating with DB server.

(9) Other interesting functions.

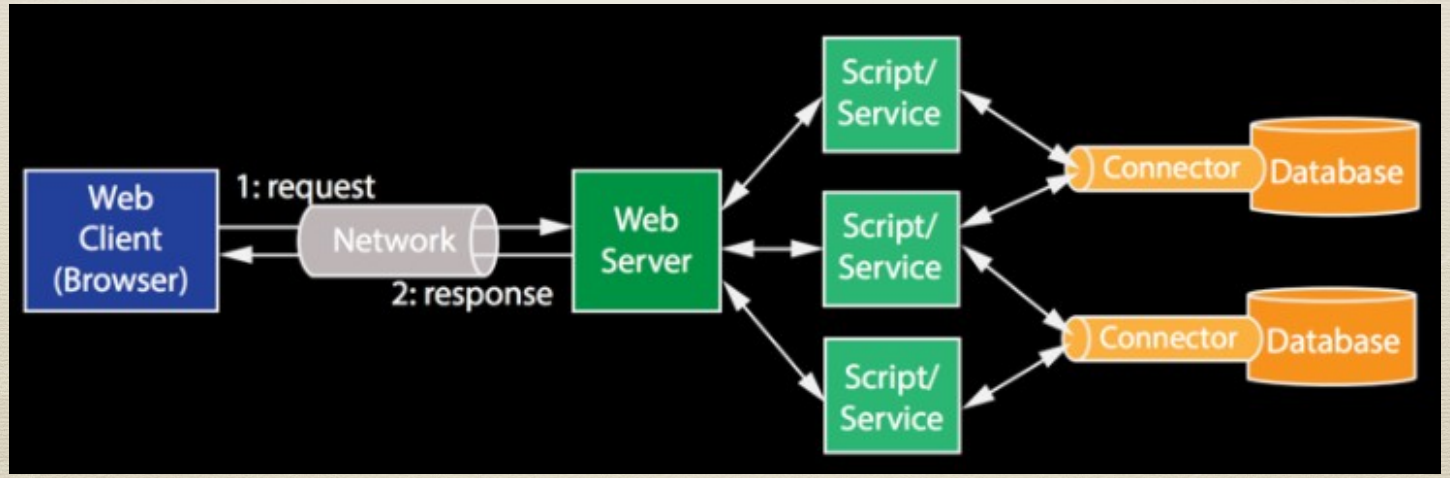
2. Start Your Project

- Recommended DBMS: https://geekflare.com/open-source-database/
- Recommended Web Framework for a beginner: Python Flask (you can use pip tool to install flask package.)
- What is web framework?
  - A web framework "is a code library that makes a developer's life easier when building reliable, scalable, and maintainable web applications" by providing reusable code or extensions for common operations.
- A Series of Tutorial for Learning Flask
  https://hackersandslackers.com/your-first-flask-application/



3. Three Layers in Web Application

- There are three essential layers in web application.
- Database Server: Create your data base first, including your primary key, foreign key, and other constraints.
- Application Layer: This is the middle layer where business and presentation logic work together to deliver the response back to the users. (Flask can help you communicate with front-end layout.)
- Front-End Layer: This layer is where technologies such as HTML ,CSS, and Javascript create the look and feel of our application.



4. Application Layer

- Taking sqlite as an example to dump your data.
- Before executing SQL queries, application layer should connect your database server first.

```
import sqlite3

conn = sqlite3.connect('./iris.db')
cursor = conn.cursor()
cursor.execute("SELECT * FROM IRIS")
data = cursor.fetchall()


for row in data:
    print (row)
```

- This following script is to implement the function of create data and search data.

```
def insert_item(sepal_length, sepal_width, petal_length, petal_width, species, cursor):
    command = "INSERT INTO IRIS (SEPAL_LENGTH, SEPAL_WIDTH, PETAL_LENGTH, \
                                 PETAL_WIDTH, SPECIES) VALUES (%f, %f, %f, %f, \'%s\');" \
    % (sepal_length, sepal_width, petal_length, petal_width, species)
    _cursor.execute(command)

def search_item(cursor):
    command = "SELECT * FROM IRIS"
    cursor.execute(commend)
    data = cursor.fetchall()

conn = sqlite3.connect('yourdb.db')
_cursor = conn.cursor()
```

5. Front-End Layer

- This is an example to launch your local server with Flask framework.
- Web Frameworks like Flask enable us to leverage Routes and Templates which make the presentation logic so much easier.

```python
from flask import Flask
app = Flask(__name__)

@app.route("/")
def hello():
    return "Hello World!"

if __name__ == "__main__":
    app.run(debug=True)
```

- Routes:
  -In Flask, conceptually, @route notify the framework about the existence of specific URLs and the function meant to handle them. Flask calls our functions that get a request and return a response views.
  -When Flask processes an HTTP request it uses this information to figure out which views it should pass the request to. The function can then return data in a variety of formats (HTML, JSON, plain text) that will be used by Flask to create an HTTP response.

```python
@app.route('/show_data', methods=['GET'])
def show_all_events():
  conn = sqlite3.connect("yourDB.sqlite3")
  cursor = conn.execute('select * from your_table')
  rows = cursor.fetchall()
  return render_template('events.html', rows = rows)
```

5. Front-End Layer (cont.)

- Templates:
  - The above example was rather simple, we could hardcode the entire HTML page inline. However, real HTML pages are often more complex, and coding contents inline is simply too tedious, error-prone, and repetitive. So, we should use templates.
  - Templates do not change what is presented to the users, but it makes the how much more organized, customizable, and extensible.

```
from flask import Flask
from flask import render_template

app = Flask(__name__)

@app.route('/')
def index():
    return render_template('abc.html')

if __name__ == '__main__':
    app.debug = True
    app.run()
```

6. HTTP Methods

```
@app.route('/show_data', methods=['GET'])
def show_all_events():
  conn = sqlite3.connect("yourDB.sqlite3")
  cursor = conn.execute('select * from your_table')
  rows = cursor.fetchall()
  return render_template('events.html', rows = rows)
```

- How does the frontend communicate with backend? -> HTTP Methods
- HTTP methods are the standard way of sending information to and from a web server. Two commonly methods are POST & GET.
- GET is typically used to retrieve information from a web server.
- POST is more often used when uploading a file, getting form data and sending sensitive data. POST is a secure way to send data to a web server.

7. File Structure

- Here is a simple example for beginners to organize your code and develop a web application.

  - template (a file directory): saving your static web pages for rendering xxx.html (index.html, insert.html, search.html, update.html, …)
  - static (a file directory): saving your web styles xxx.css, xxx.js
  - app.py (main program): (application layer: for communication)
  - xxx.db

```
[13:17 Heng [~/Data_Base/final_project_tutorial/flask_hello_world] $ ls
app.py    templates
[13:17 Heng [~/Data_Base/final_project_tutorial/flask_hello_world] $ python app.py
 * Serving Flask app "app" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 316-553-152
```

Launch your local server by executing app.py

8. Example: DB Connection

The following section is an example developed with PostgreSQL and Flask framework for beginners. And, it is only part of our program.

After create DB table, connect to your DB server.

(Write in your app.py)

```
app = Flask(__name__)

# connect to Database
POSTGRES = {
    'user':           ,
    'pw': 'root',
    'db': 'Student',
    'host': 'localhost',
    'port': '5432'
}
app.config['SQLALCHEMY_DATABASE_URI'] = 'postgresql://%(user)s:\%(pw)s@%(host)s:%(port)s/%(db)s' % POSTGRES
db = SQLAlchemy(app)
```

8. Example: Front-End Layer

Render your template with html file. So, you have five pages which are homepages of system, insertion, deletion, searching, and modification, respectively.

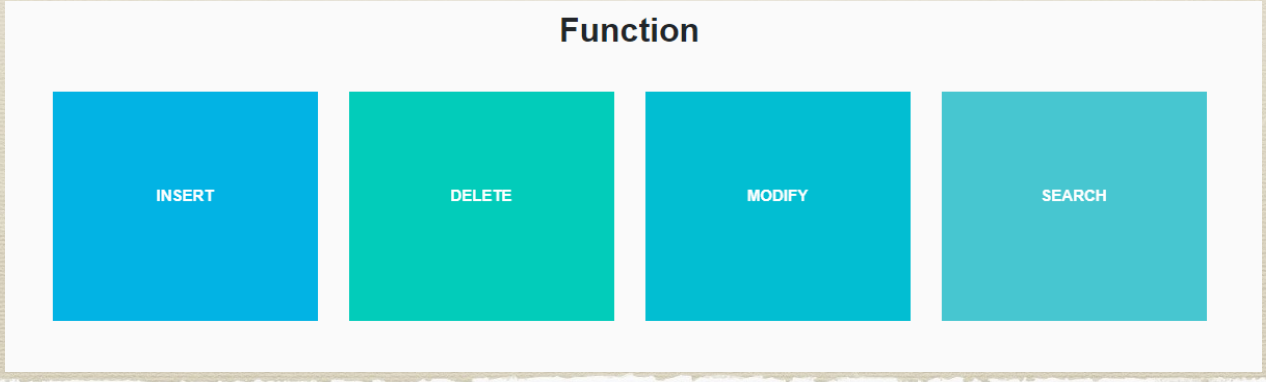
```python
# render index.html template
@app.route('/')
def index():
    return render_template('index.html')

# render search.html template
@app.route('/search')
def search():
    return render_template('search.html')

# render delete.html template
@app.route('/delete')
def delete():
    return render_template('delete.html')

# render modify.html template
@app.route('/modify')
def modify():
    return render_template('modify.html')

# render insert.html template
@app.route('/insert')
def insert():
    return render_template('add.html')
```



8. Example: CRUD Function

Take deletion function as an example. The following code can be used to delete data and get the data table.

Delete your data

```
# get delete_data form, delete the id's data then direct to search.html
@app.route('/delete_data',methods=['POST'])
def delete_data():
    if request.method == 'POST':
        # get form data
        table_name = request.form.get('table_name')
        if table_name == 'grade':
            student_id  = request.form.get('student_id')
            course_id = request.form.get('course_id')
            sql = "DELETE FROM %s WHERE student_id ='%s' and course_id = '%s'" %(table_name ,student_id ,course_id)
        elif table_name == 'course':
            course_id   = request.form.get('course_id')
            sql = "DELETE FROM %s WHERE course_id='%s'" %(table_name ,course_id)
        elif table_name == 'student':
            student_id  = request.form.get('student_id')
            sql = "DELETE FROM %s WHERE student_id='%s'" %(table_name ,student_id)
        # execute the SQL and commit to db
        db.session.execute(sql)
        db.session.commit()
        results = db.session.execute("select * from %s" %(table_name))
        return render_template('search.html', output_data = results.fetchall() ,title = results.keys())
```

Show the data table after deletion

```
# get delete_get_data form and return the column name
@app.route('/delete_get_data',methods=['POST'])
def delete_get_data():
    if request.method == 'POST':
        # get form data
        table_name = request.form.get('table_name')
        # execute the SQL
        results = db.session.execute("select * from %s" %(table_name))
        return render_template('delete.html', output_data = results.fetchall() ,title = results.keys() ,table_name = table_name)
```

8. Example: CRUD Function

Take insertion function as an example. The following code can be used to insert value and get the data table.

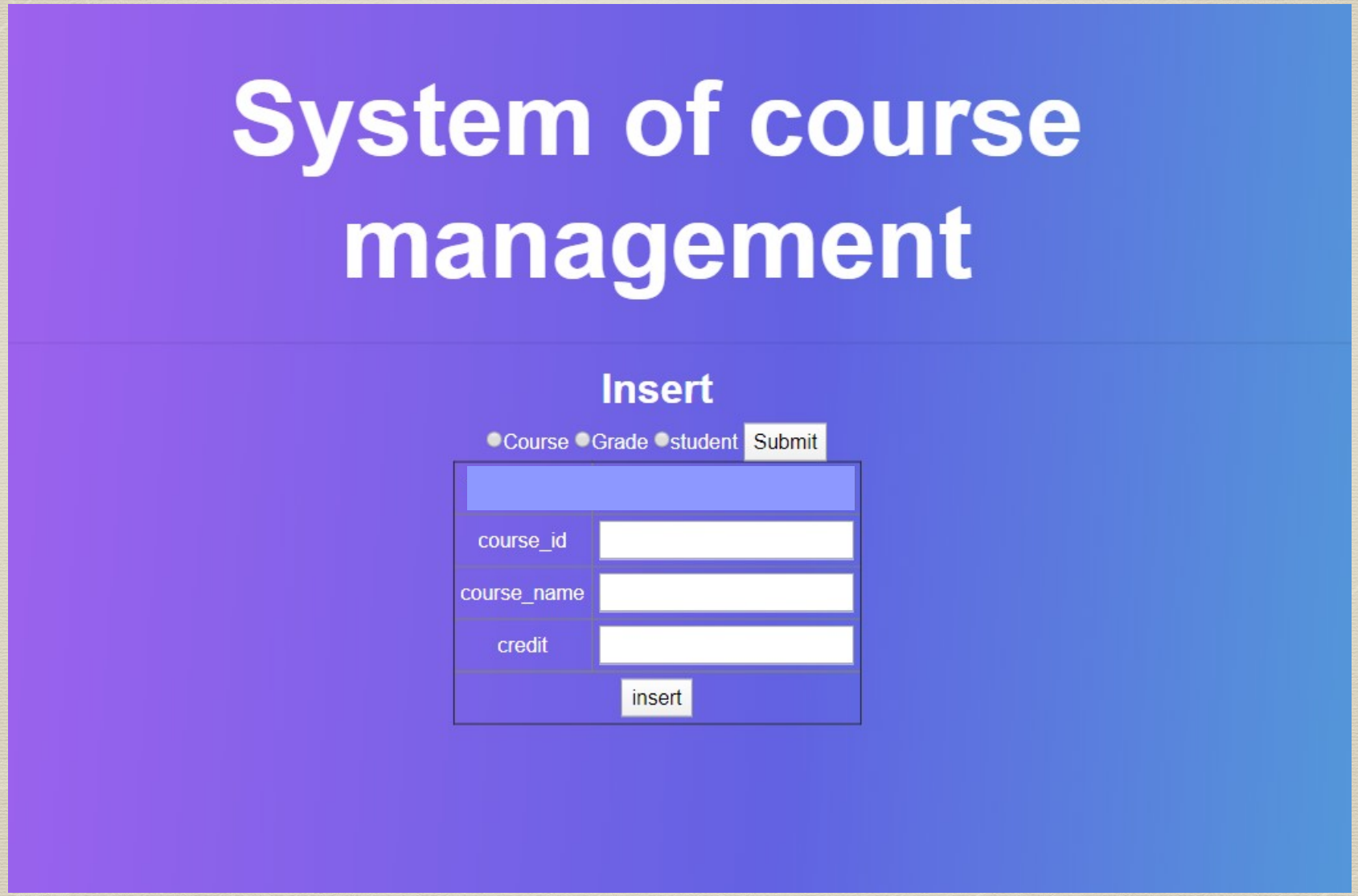
Insert your data

```
# get insert_data form and insert the data to db, then direct to search.html
@app.route('/insert_data',methods=['POST'])
def insert_data():
    if request.method == 'POST':
        table_name = request.form.get('table_name')
        if table_name == 'grade':
            # get form data
            student_id  = request.form.get('student_id')
            course_id = request.form.get('course_id')
            score = request.form.get('score')
            sql = "INSERT INTO %s (student_id, course_id, score) VALUES ('%s','%s','%s');" \
            %(table_name ,student_id ,course_id ,score)
        elif table_name == 'course':
            # get form data
            course_id   = request.form.get('course_id')
            course_name = request.form.get('course_name')
            credit = request.form.get('credit')
            sql = "INSERT INTO %s (course_id, course_name, credit) VALUES ('%s','%s','%s');" \
            %(table_name ,course_id ,course_name ,credit)
        elif table_name == 'student':
            # get form data
            student_id  = request.form.get('student_id')
            student_name = request.form.get('student_name')
            gender = request.form.get('gender')
            birthday = request.form.get('birthday')
            fruit = request.form.get('fruit')
            sql = "INSERT INTO %s (student_name, gender, birthday,fruit) VALUES ('%s','%s','%s','%s','%s');" \
            %(table_name ,student_id ,student_name ,gender,birthday,fruit)
        # execute the SQL and commit to db
        db.session.execute(sql)
        db.session.commit()
        results = db.session.execute("select * from %s" %(table_name))
        return render_template('search.html', output_data = results.fetchall() ,title = results.keys())
```

Show the data table after insertion

```
# get insert_get_data form and return the column name
@app.route('/insert_get_data',methods=['POST'])
def insert_get_data():
    if request.method == 'POST':
        # get form data
        table_name = request.form.get('table_name')
        # execute the SQL
        results = db.session.execute("select * from %s" %(table_name))
        return render_template('add.html', output_data = results.fetchall() ,title = results.keys() ,table_name = table_name)
```

8. Example: UI


System of course management
Insert
Course
Grade
student
Submit
course_id
course_name
credit
insert