웹프로그래밍의 기초

Week11

WTF and DBMS

Forms

Intro. HTML Forms

Web forms, such as text fields and text areas, give users the ability to send data to your application.

- HTML Forms
  - An HTML form is used to collect user input. The user input is most often sent to a server for processing.
- The <form> Element
  - The HTML <form> element is used to create an HTML form for user input:

    <form>

    .form elements

    .</form>
  - The <form> element is a container for different types of input elements, such as: text fields, checkboxes, radio buttons, submit buttons, etc.
- The <input> Element
  - The HTML <input> element is the most used form element.

Intro. HTML Forms - example

```html
<!DOCTYPE html>
<html>
<body>

<h2>Text input fields</h2>

<form>
  <label for="fname">First name:</label><br>
  <input type="text" id="fname" name="fname" value="John"><br>
  <label for="lname">Last name:</label><br>
  <input type="text" id="lname" name="lname" value="Doe">
</form>

<p>Note that the form itself is not visible.</p>

<p>Also note that the default width of text input fields is 20
characters.</p>

</body>
</html>
```

Text input fields

First name:

John

Last name:

Doe

Note that the form itself is not visible.

Also note that the default width of text input fields is 20 characters.

Flask-WTF

- To render and validate web forms in a safe and flexible way in Flask, you'll use Flask-WTF, which is a Flask extension that helps you use the WTForms library in your Flask application.

Table 4-1. WTForms standard HTML fields

Field type	Description
`StringField`	Text field
`TextAreaField`	Multiple-line text field
`PasswordField`	Password text field
`HiddenField`	Hidden text field
`DateField`	Text field that accepts a `datetime.date` value in a given format
`DateTimeField`	Text field that accepts a `datetime.datetime` value in a given format
`IntegerField`	Text field that accepts an integer value
`DecimalField`	Text field that accepts a `decimal.Decimal` value
`FloatField`	Text field that accepts a floating-point value
`BooleanField`	Checkbox with `True` and `False` values
`RadioField`	List of radio buttons
`SelectField`	Drop-down list of choices
`SelectMultipleField`	Drop-down list of choices with multiple selection
`FileField`	File upload field
`SubmitField`	Form submission button
`FormField`	Embed a form as a field in a container form
`FieldList`	List of fields of a given type

Table 4-2. WTForms validators

Validator	Description
`Email`	Validates an email address
`EqualTo`	Compares the values of two fields; useful when requesting a password to be entered twice for confirmation
`IPAddress`	Validates an IPv4 network address
`Length`	Validates the length of the string entered
`NumberRange`	Validates that the value entered is within a numeric range
`Optional`	Allows empty input on the field, skipping additional validators
`Required`	Validates that the field contains data
`Regexp`	Validates the input against a regular expression
`URL`	Validates a URL
`AnyOf`	Validates that the input is one of a list of possible values
`NoneOf`	Validates that the input is none of a list of possible values

Learn by example

- To build a web form, you will create a subclass of the FlaskForm base class, which you import from the flask_wtf package. You also need to specify the fields you use in your form, which you will import from the wtforms package.
- Let's say we are now building a web application for course creation. Then we might think of the following four items.
  - **Title**: A text input field for the course title.

    -> StringField
  - **Description**: A text area field for the course description.

    -> TelField
  - **Level**: A radio field for the course level with three choices: 1학년, 2학년, 3학년, and 4학년

    -> RadioField
  - **Available**: A checkbox field that indicates whether the course is currently available(개설).

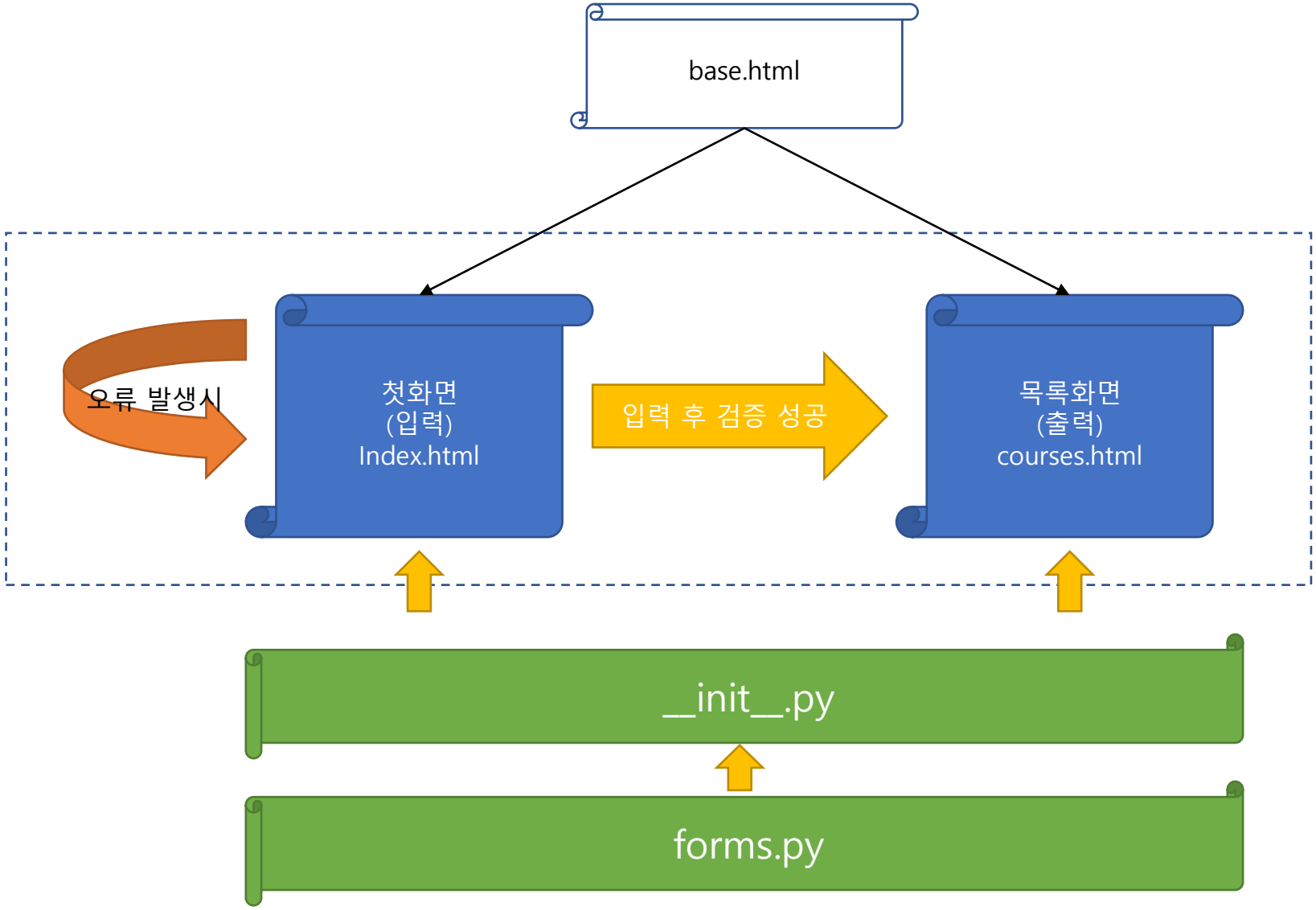
    -> BooleanField
- To make sure every field is not empty, and to control the length of some important fields, we will use two validators – InputRequired and Length.

```
~/projects/webp/app
 |
 └──── forms.py
```

```
from flask_wtf import FlaskForm
from wtforms import (StringField, TextAreaField, RadioField, BooleanField)
from wtforms.validators import InputRequired, Length

class CourseForm(FlaskForm):
    title = StringField('과목이름', validators=[InputRequired(), Length(min=10, max=100)])
    description = TextAreaField('과목설명', validators=[InputRequired(), Length(max=200)])
    level = RadioField('권장학년', choices=['1학년','2학년','3학년','4학년'],validators=[InputRequired()])
    available = BooleanField('개설', default='checked')
```

Learn by example – Logic Flow

```
~/projects₩webp
│
└──── app/
      ├── __init__.py
      ├── forms.py
      └── templates/
            ├── base.html
            ├── index.html
            └── courses.html
```



```
~/projects/webp/app
 |
 └──── __init__.py
```

```python
from flask import Flask, render_template, redirect, url_for
from .forms import CourseForm

app = Flask(__name__)
app.config['SECRET_KEY'] = 'your secret key'


courses_list = [{
    'title': '웹프로그래밍의 기초',
    'description': 'Python 프로그래밍의 기초를 이해하고 Flask를 이용한 백엔드 웹 애플리케이션 개발에 필요한 개념을 연구하고 실제로 실습함으로써, 문헌정보학 전공자(사서)로서 지녀야 할 동적 웹페이지 시스템에 대한 이해도를 높인다.',
    'available': True,
    'level': '3학년'
    }]


@app.route('/', methods=('GET', 'POST'))
def index():
    form = CourseForm()
    if form.validate_on_submit():
        courses_list.append({'title': form.title.data,
                             'description': form.description.data,
                             'available': form.available.data,
                             'level': form.level.data
                             })
        return redirect(url_for('courses'))
    return render_template('index.html', form=form)


@app.route('/courses/')
def courses():
    return render_template('courses.html', courses_list=courses_list)
```

~/projects/webp/app/templates
│
└── base.html

```
<!DOCTYPE html>
<html lang="UTF-8">
<head>
    <meta charset="UTF-8">
    <title>{% block title %}{% endblock %} - FlaskApp</title>
    <style>
        nav a {
            color: #ff0000;
            font-size: 2em;
        }
    </style>
</head>
<body>
    <nav>
        <a href="{{ url_for('index') }}">[과목 추가]</a>
        <a href="{{ url_for('courses') }}">[과목 목록]</a>
    </nav>
    <hr>
    <div class="content">
        {% block content %} {% endblock %}
    </div>
</body>
</html>
```

~/projects/webp/app/templates
│
└── courses.html

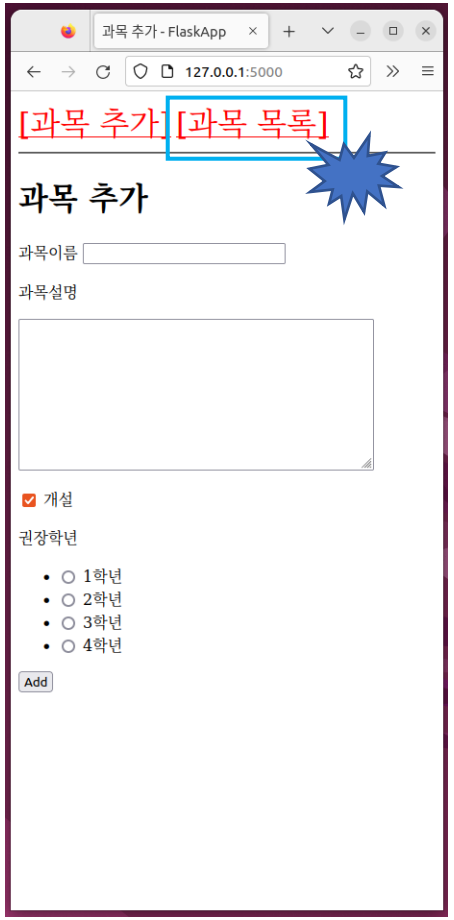
```
{% extends 'base.html' %}

{% block content %}
    <h1>{% block title %}과목 목록{% endblock %}</h1>
    <hr>
    {% for course in courses_list %}
        <h2> {{ course['title'] }} </h2>
        <h4> {{ course['description'] }} </h4>
        <p><i>권장학년({{ course['level'] }})</i></p>
        <p>개설여부:
            {% if course['available'] %}
                                    개설
            {% else %}
                                    미개설
            {% endif %}</p>
        <hr>
    {% endfor %}
{% endblock %}
```

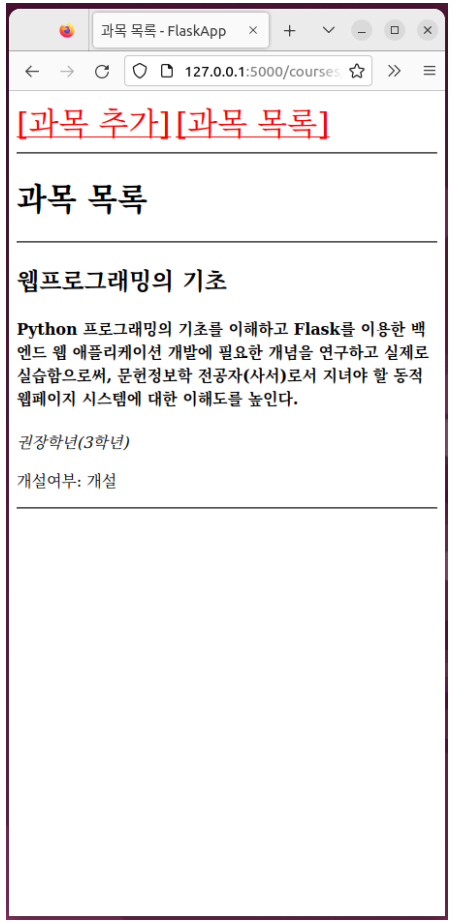
```
~/projects/webp/app/templates
│
└──── index.html
```

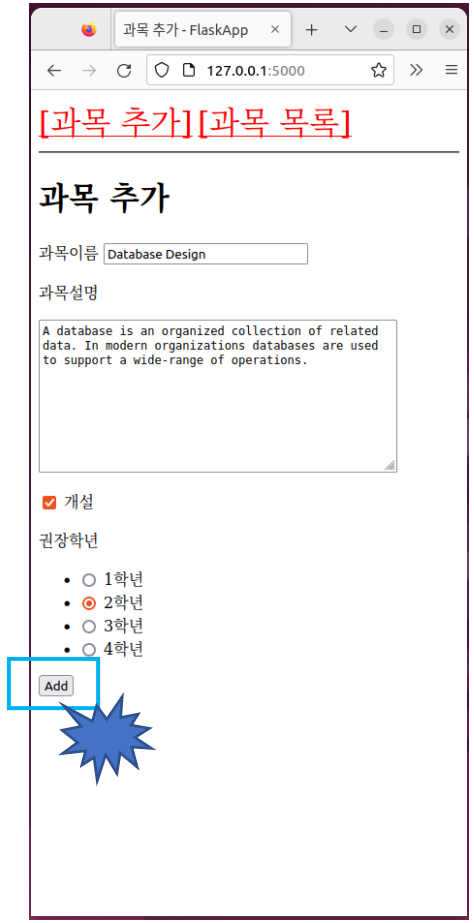
```
{% extends 'base.html' %}

{% block content %}
    <h1>{% block title %}과목 추가{% endblock %}</h1>

    <form method="POST" action="/">
        {{ form.csrf_token }}
        <p>
            {{ form.title.label }}
            {{ form.title(size=20) }}
        </p>

        {% if form.title.errors %}
            <ul class="errors">
                {% for error in form.title.errors %}
                    <li>{{ error }}</li>
                {% endfor %}
            </ul>
        {% endif %}

        <p>
            {{ form.description.label }}
        </p>
        {{ form.description(rows=10, cols=50) }}

        {% if form.description.errors %}
            <ul class="errors">
                {% for error in form.description.errors %}
                    <li>{{ error }}</li>
                {% endfor %}
            </ul>
        {% endif %}
```

```
        <p>
            {{ form.available() }} {{ form.available.label }}
        </p>

        {% if form.available.errors %}
            <ul class="errors">
                {% for error in form.available.errors %}
                    <li>{{ error }}</li>
                {% endfor %}
            </ul>
        {% endif %}

        <p>
            {{ form.level.label }}
            {{ form.level() }}
        </p>

        {% if form.level.errors %}
            <ul class="errors">
                {% for error in form.level.errors %}
                    <li>{{ error }}</li>
                {% endfor %}
            </ul>
        {% endif %}

        <p>
            <input type="submit" value="Add">
        </p>
    </form>

{% endblock %}
```

Learn by example – Results

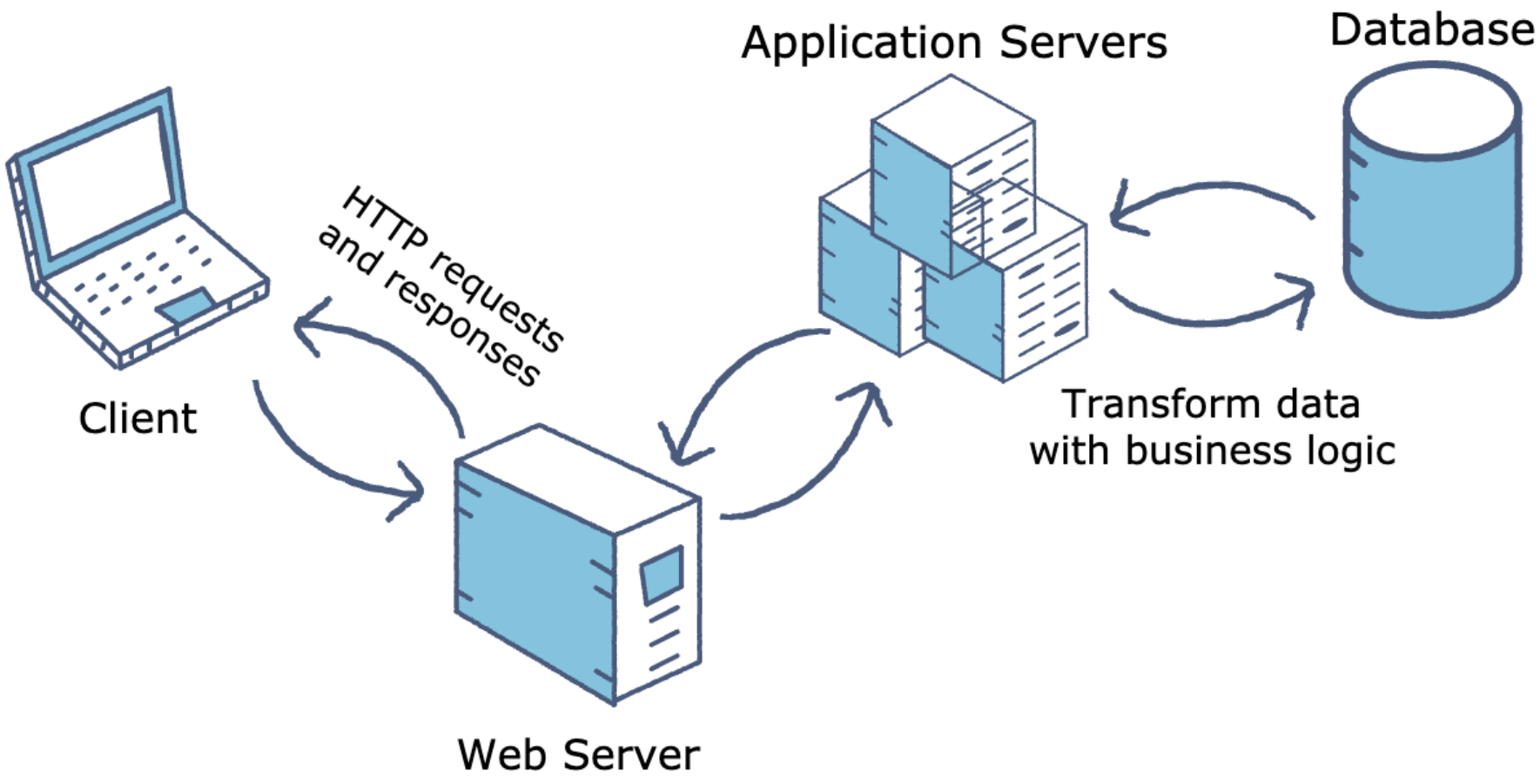






Introduction to Database

Application Server and Database Architecture



Definitions

Database ⇒

- Collection of related data 관련된 데이터 and its metadata
  organized in a structured format 구조적 형식
- for optimized information management 정보 관리

Database Management System (DBMS) ⇒

- Software that enables
  easy creation 구축, modification 변경, & access 접속 of databases
- for efficient and effective database management 데이터베이스 관리

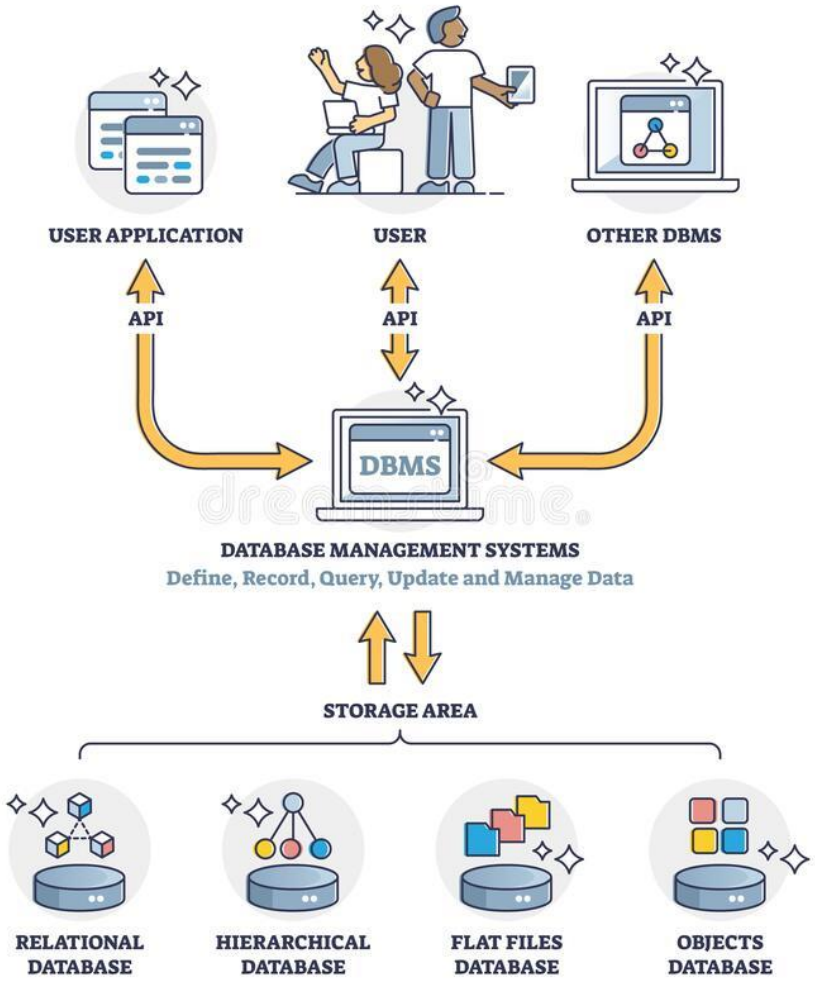
Database System ⇒

- Integrated system 통합 시스템 of
  hardware, software, data, procedures, & people
- that define 결정 and regulate 규제
  the collection, storage, management, & use of data
  within a database environment 데이터베이스 환경

Database Management System (DBMS)

- A database management system (DBMS) is software that controls the storage, organization, and retrieval of data. Typically, a DBMS has the following elements:
  - Kernel code
    - This code manages memory and storage for the DBMS.
  - Repository of metadata
    - This repository is usually called a data dictionary.
  - Query language
    - This language enables applications to access the data.
- A database application is a software program that interacts with a database to access and manipulate data.

Database Management System (DBMS) – Con’td



SQL definition from Oracle

- **A query, or SQL SELECT** statement, selects data from one or more tables or views.

```
SELECT select_list FROM source_list
```

- **Data Manipulation Language (DML)** statements add, change, and delete Oracle Database table data. A transaction is a sequence of one or more SQL statements that Oracle Database treats as a unit: either all of the statements are performed, or none of them are.

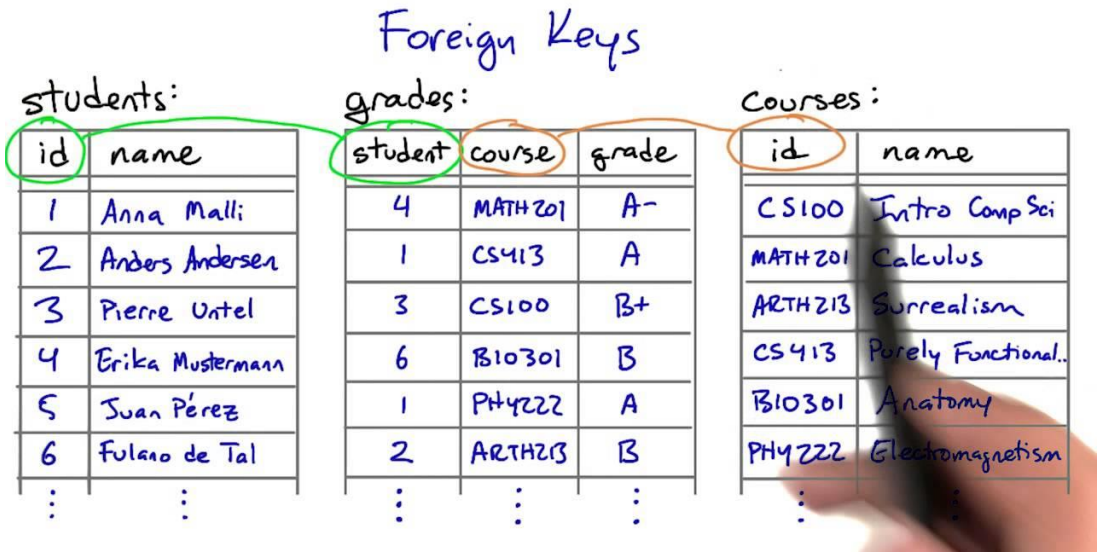
```
INSERT INTO table_name (list_of_columns)
VALUES (list_of_values);

UPDATE table_name
SET column_name = value [, column_name = value]...
[ WHERE condition ];

DELETE FROM table_name [ WHERE condition ];
```

RDB basics

- A table is a collection of related data held in a table format within a database. It consists of columns and rows.

- The UNIQUE constraint ensures that all values in a column are different.
- Both the UNIQUE and PRIMARY KEY constraints provide a guarantee for uniqueness for a column or set of columns.
- A PRIMARY KEY constraint automatically has a UNIQUE constraint.
- However, you can have many UNIQUE constraints per table, but only one PRIMARY KEY constraint per table.

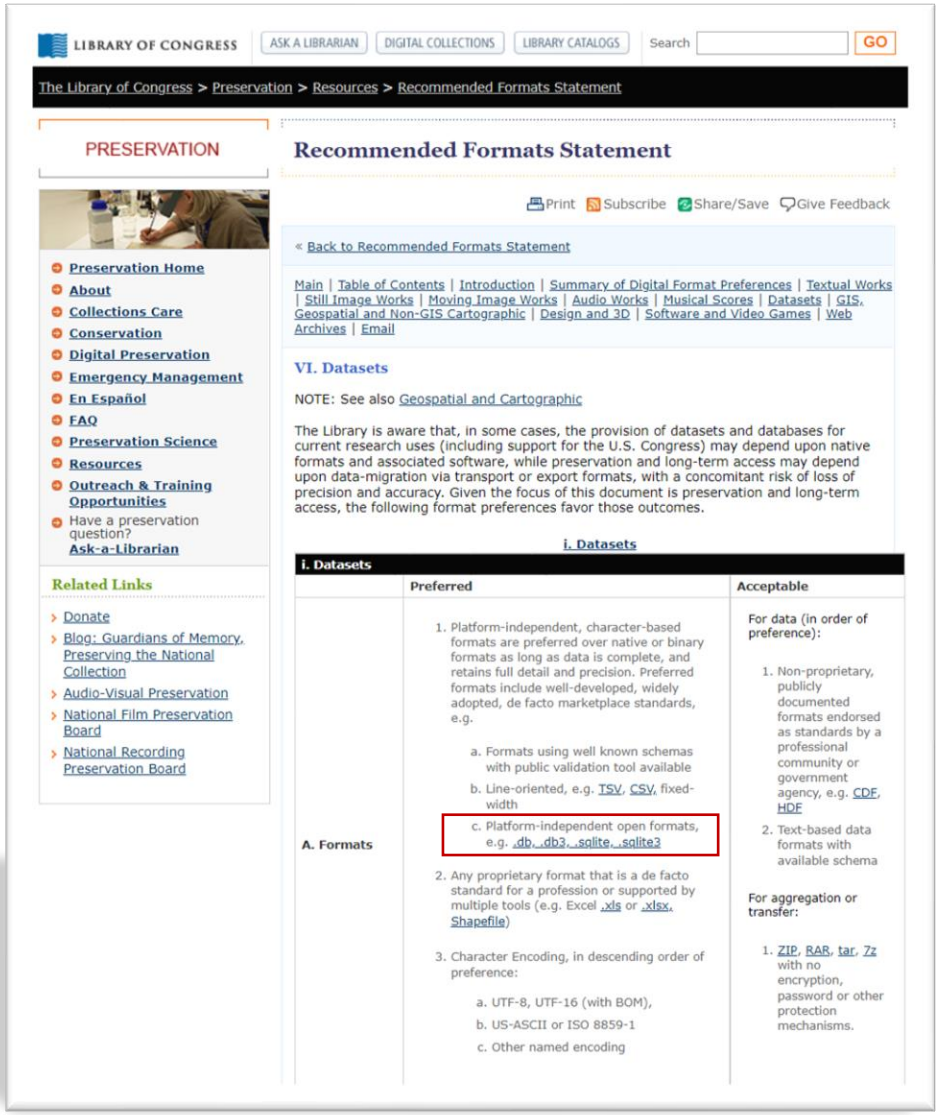
- A FOREIGN KEY is a field (or collection of fields) in one table, that refers to the PRIMARY KEY in another table.
- The table with the foreign key is called the child table, and the table with the primary key is called the referenced or parent table.



Introduction to SQLite and SQLAlchemy

SQLite

- D. Richard Hipp was originally designing software used for a damage-control system aboard guided-missile destroyers, in August 2000, version 1.0 of SQLite was released, with storage based on gdbm (GNU Database Manager).
- SQLite is a database engine written in the C programming language. It is not a standalone app; rather, it is a library that software developers embed in their apps. As such, it belongs to the family of embedded databases.
- SQLite is a Recommended Storage Format for datasets according to the US Library of Congress.



LIBRARY OF CONGRESS | ASK A LIBRARIAN | DIGITAL COLLECTIONS | LIBRARY CATALOGS | Search | GO

The Library of Congress > Preservation > Resources > Recommended Formats Statement

PRESERVATION

- Preservation Home
- About
- Collections Care
- Conservation
- Digital Preservation
- Emergency Management
- En Español
- FAQ
- Preservation Science
- Resources
- Outreach & Training Opportunities
- Have a preservation question? Ask-a-Librarian

Related Links

- Donate
- Blog: Guardians of Memory, Preserving the National Collection
- Audio-Visual Preservation
- National Film Preservation Board
- National Recording Preservation Board

Recommended Formats Statement

Print | Subscribe | Share/Save | Give Feedback

« Back to Recommended Formats Statement

Main | Table of Contents | Introduction | Summary of Digital Format Preferences | Textual Works | Still Image Works | Moving Image Works | Audio Works | Musical Scores | Datasets | GIS, Geospatial and Non-GIS Cartographic | Design and 3D | Software and Video Games | Web Archives | Email

VI. Datasets

NOTE: See also Geospatial and Cartographic

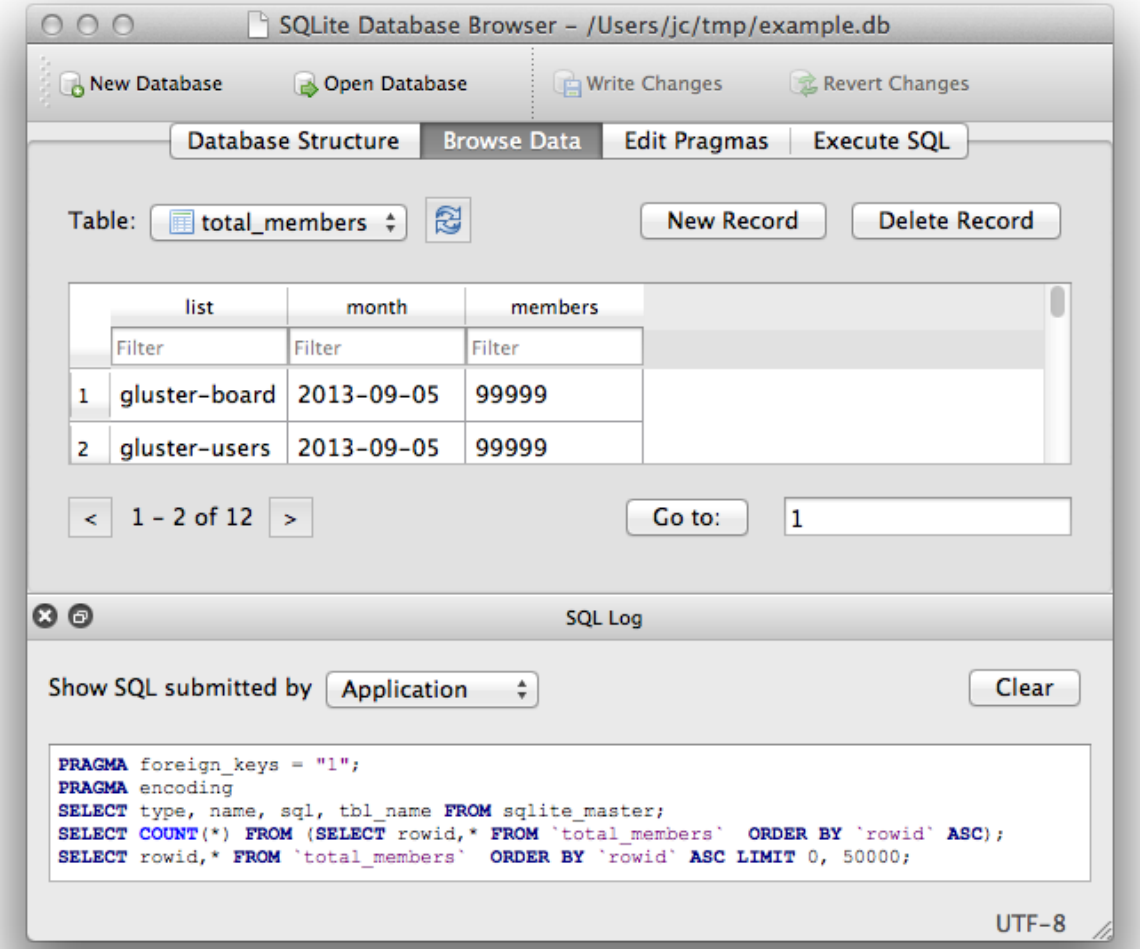
The Library is aware that, in some cases, the provision of datasets and databases for current research uses (including support for the U.S. Congress) may depend upon native formats and associated software, while preservation and long-term access may depend upon data-migration via transport or export formats, with a concomitant risk of loss of precision and accuracy. Given the focus of this document is preservation and long-term access, the following format preferences favor those outcomes.

i. Datasets

i. Datasets	Preferred	Acceptable
A. Formats	1. Platform-independent, character-based formats are preferred over native or binary formats as long as data is complete, and retains full detail and precision. Preferred formats include well-developed, widely adopted, de facto marketplace standards, e.g. a. Formats using well known schemas with public validation tool available b. Line-oriented, e.g. TSV, CSV, fixed-width c. Platform-independent open formats, e.g. .db, .db3, .sqlite, .sqlite3 2. Any proprietary format that is a de facto standard for a profession or supported by multiple tools (e.g. Excel .xls or .xlsx, Shapefile) 3. Character Encoding, in descending order of preference: a. UTF-8, UTF-16 (with BOM), b. US-ASCII or ISO 8859-1 c. Other named encoding	For data (in order of preference): 1. Non-proprietary, publicly documented formats endorsed as standards by a professional community or government agency, e.g. CDF, HDF 2. Text-based data formats with available schema For aggregation or transfer: 1. ZIP, RAR, tar, 7z with no encryption, password or other protection mechanisms.

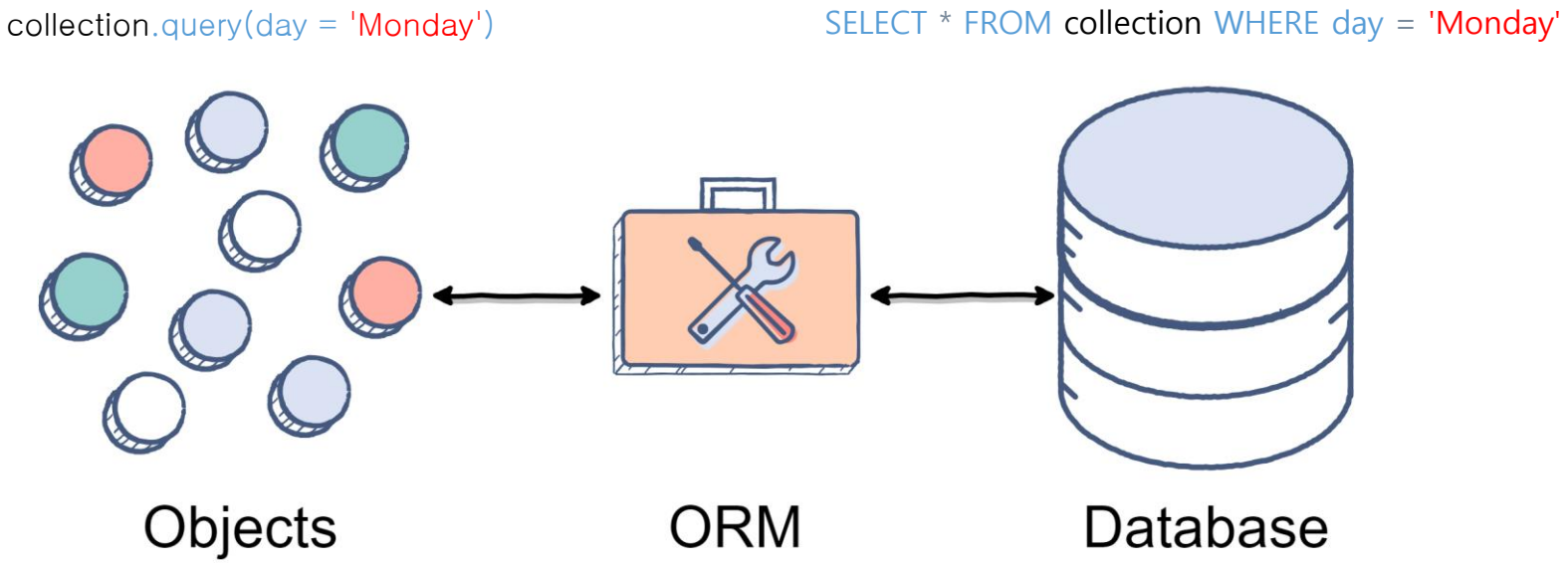
Easy way to access to SQLite database

- DB Browser for SQLite (DB4S) is a high quality, visual, open source tool to create, design, and edit database files compatible with SQLite.
- This program is not a visual shell for the sqlite command line tool, and does not require familiarity with SQL commands.



ORM

- Object Relational Mapping (ORM) is a technique used in creating a "bridge" between object-oriented programs and, in most cases, relational databases.
- Popular ORM Tools for Python are Django, SQLObject, and SQLAlchemy.



Pros & Cons of ORM

- Advantages of Using ORM Tools
  - It speeds up development time for teams.
  - Decreases the cost of development.
  - Handles the logic required to interact with databases.
  - Improves security. ORM tools are built to eliminate the possibility of SQL injection attacks.
  - You write less code when using ORM tools than with SQL.
- Disadvantages of Using ORM Tools
  - Learning how to use ORM tools can be time consuming.
  - They are likely not going to perform better when very complex queries are involved.
  - ORMs are generally slower than using SQL.

ORM example

SQL create table statement

```
CREATE TABLE students (
   id INTEGER NOT NULL,
   name VARCHAR,
   lastname VARCHAR,
   PRIMARY KEY (id)
)
```

SQLAlchemy table creation code

```
from sqlalchemy import create_engine, MetaData, Table, Column, Integer,
String
engine = create_engine('sqlite:///college.db', echo = True)
meta = MetaData()

students = Table(
   'students', meta,
   Column('id', Integer, primary_key = True),
   Column('name', String),
   Column('lastname', String),
)
meta.create_all(engine)
```

SQLAlchemy

- SQLAlchemy is an SQL toolkit that provides efficient and high-performing database access for relational databases.
- It is contained in Flask-Migrate package.
- It also gives you an Object Relational Mapper (ORM), which allows you to make queries and handle data using simple Python objects and methods.
- Flask-SQLAlchemy is a Flask extension that makes using SQLAlchemy with Flask easier, providing you tools and methods to interact with your database in your Flask applications through SQLAlchemy.

Building a simple Q&A board
Step 1. Database

Database Configuration

- With creating config.py file, you can choose where to store the database file.

```
~/projects/webp
 |
 └──── config.py
```

```
import os

BASE_DIR = os.path.dirname(__file__)

SQLALCHEMY_DATABASE_URI = 'sqlite:///{}'.format(os.path.join(BASE_DIR, 'webp.db'))
SQLALCHEMY_TRACK_MODIFICATIONS = False
```

Create repository for your database

- Build '__init__.py' file as follows

```
~/projects/webp/app
 │
 └──── __init__.py
```

```
from flask import Flask
from flask_migrate import Migrate
from flask_sqlalchemy import SQLAlchemy

import config

db = SQLAlchemy()
migrate = Migrate()

def create_app():
    app = Flask(__name__)
    app.config.from_object(config)

    # ORM
    db.init_app(app)
    migrate.init_app(app, db)

    return app
```

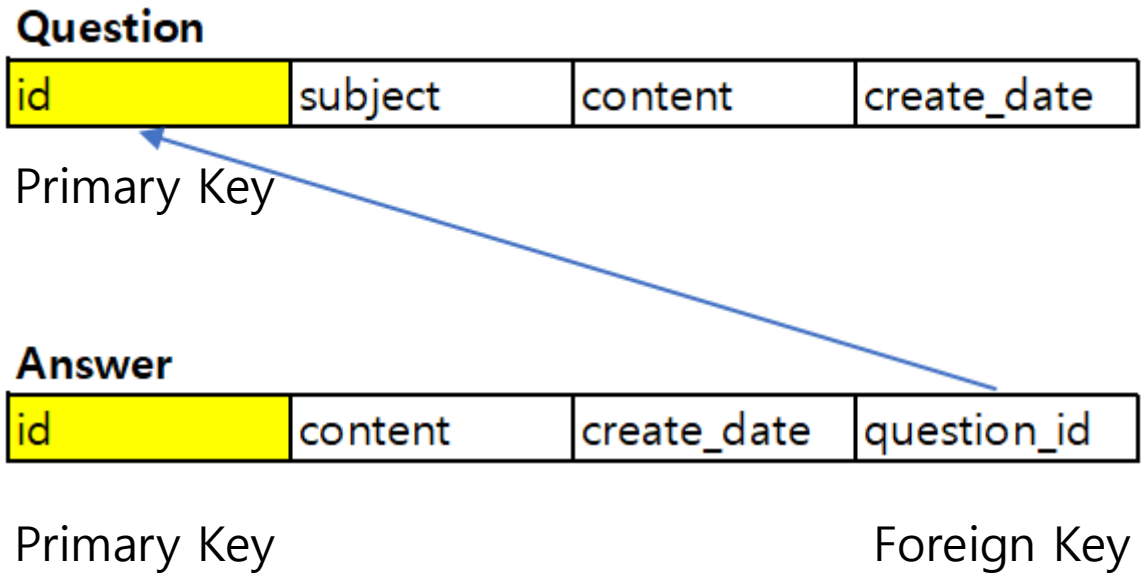
Create repository for your database (Cont'd)

- Run ‘flask db init’ to initiate your SQLite database in webp.

```
scott@scott-virtual-machine: ~/projects/webp
(webp) scott@scott-virtual-machine:~/projects/webp$ ls -al app/
total 16
drwxrwxr-x 3 scott scott 4096 11월  7 00:31 .
drwxrwxr-x 4 scott scott 4096 11월  7 00:31 ..
-rw-rw-r-- 1 scott scott  311 11월  7 00:31 __init__.py
drwxrwxr-x 2 scott scott 4096 11월  7 00:31 __pycache__
(webp) scott@scott-virtual-machine:~/projects/webp$ ls -al
total 20
drwxrwxr-x 4 scott scott 4096 11월  7 00:31 .
drwxrwxr-x 4 scott scott 4096 11월  7 00:26 ..
drwxrwxr-x 3 scott scott 4096 11월  7 00:31 app
-rw-rw-r-- 1 scott scott  171 11월  3 03:05 config.py
drwxrwxr-x 2 scott scott 4096 11월  7 00:31 __pycache__
(webp) scott@scott-virtual-machine:~/projects/webp$ flask db init
  Creating directory /home/scott/projects/webp/migrations ...  done
  Creating directory /home/scott/projects/webp/migrations/versions ...  done
  Generating /home/scott/projects/webp/migrations/script.py.mako ...  done
  Generating /home/scott/projects/webp/migrations/env.py ...  done
  Generating /home/scott/projects/webp/migrations/alembic.ini ...  done
  Generating /home/scott/projects/webp/migrations/README ...  done
  Please edit configuration/connection/logging settings in '/home/scott/projects/webp/migrations/alembic.ini' before proceeding.
(webp) scott@scott-virtual-machine:~/projects/webp$
```

Database design

- For Q&A board, we need two tables linked together as follows.



Declare Models

- we define module-level constructs that will form the structures which we will be querying from the database. This structure, known as a Declarative Mapping, defines at once **both a Python object model, as well as database metadata** that describes real SQL tables that exist, or will exist, in a particular database:

```
~/projects/webp/app
 |
 └──── __init__.py
```

```python
from flask import Flask
from flask_migrate import Migrate
from flask_sqlalchemy import SQLAlchemy

import config

db = SQLAlchemy()
migrate = Migrate()
from . import models

def create_app():
    app = Flask(__name__)
    app.config.from_object(config)

    # ORM
    db.init_app(app)
    migrate.init_app(app, db)

    return app
```

```
~/projects/webp/app
 |
 └──── models.py
```

```python
from app import db

class Question(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    subject = db.Column(db.String(200), nullable=False)
    content = db.Column(db.Text(), nullable=False)
    create_date = db.Column(db.DateTime(), nullable=False)

class Answer(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    question_id = db.Column(db.Integer, db.ForeignKey('question.id', ondelete='CASCADE'))
    question = db.relationship('Question', backref=db.backref('answer_set', cascade='all, delete-orphan'))
    content = db.Column(db.Text(), nullable=False)
    create_date = db.Column(db.DateTime(), nullable=False)
```

Create table (1. generate revision scripts)

- Revision scripts
  - Alembic provides for the creation, management, and invocation of change management scripts for a relational database, using SQLAlchemy as the underlying engine.
- Run ‘flask db migrate’ to generate revision files.

```
scott@scott-virtual-machine: ~/projects/webp
(webp) scott@scott-virtual-machine:~/projects/webp$ ls -al
total 20
drwxrwxr-x 4 scott scott 4096 11월  7 02:11 .
drwxrwxr-x 4 scott scott 4096 11월  7 00:26 ..
drwxrwxr-x 2 scott scott 4096 11월  7 02:10 app
-rw-rw-r-- 1 scott scott  171 11월  3 03:05 config.py
drwxrwxr-x 3 scott scott 4096 11월  7 00:32 migrations
(webp) scott@scott-virtual-machine:~/projects/webp$ ls -al app/
total 16
drwxrwxr-x 2 scott scott 4096 11월  7 02:10 .
drwxrwxr-x 4 scott scott 4096 11월  7 02:11 ..
-rw-rw-r-- 1 scott scott  332 11월  7 02:10 __init__.py
-rw-rw-r-- 1 scott scott  642 11월  7 02:10 models.py
(webp) scott@scott-virtual-machine:~/projects/webp$ flask db migrate
INFO  [alembic.runtime.migration] Context impl SQLiteImpl.
INFO  [alembic.runtime.migration] Will assume non-transactional DDL.
INFO  [alembic.autogenerate.compare] Detected added table 'question'
INFO  [alembic.autogenerate.compare] Detected added table 'answer'
  Generating /home/scott/projects/webp/migrations/versions/a9dca3de4599_.py ...
 done
(webp) scott@scott-virtual-machine:~/projects/webp$
```

scott@scott-virtual-machine: ~/projects/webp/migrations/v...

```
(webp) scott@scott-virtual-machine:~/projects/webp/migrations/versions$ pwd
/home/scott/projects/webp/migrations/versions
(webp) scott@scott-virtual-machine:~/projects/webp/migrations/versions$ ls -al
total 16
drwxrwxr-x 3 scott scott 4096 11월  7 02:12 .
drwxrwxr-x 4 scott scott 4096 11월  7 02:12 ..
-rw-rw-r-- 1 scott scott 1250 11월  7 02:12 a9dca3de4599_.py
drwxrwxr-x 2 scott scott 4096 11월  7 02:12 __pycache__
(webp) scott@scott-virtual-machine:~/projects/webp/migrations/versions$
```

Open | a9dca3de4599_.py ~/projects/webp/migrations/versions | Save

```python
"""empty message

Revision ID: a9dca3de4599
Revises:
Create Date: 2022-11-07 02:12:11.299682

"""
from alembic import op
import sqlalchemy as sa


# revision identifiers, used by Alembic.
revision = 'a9dca3de4599'
down_revision = None
branch_labels = None
depends_on = None


def upgrade():
    # ### commands auto generated by Alembic - please adjust! ###
    op.create_table('question',
    sa.Column('id', sa.Integer(), nullable=False),
    sa.Column('subject', sa.String(length=200), nullable=False),
    sa.Column('content', sa.Text(), nullable=False),
    sa.Column('create_date', sa.DateTime(), nullable=False),
    sa.PrimaryKeyConstraint('id')
    )
    op.create_table('answer',
    sa.Column('id', sa.Integer(), nullable=False),
    sa.Column('question_id', sa.Integer(), nullable=True),
    sa.Column('content', sa.Text(), nullable=False),
    sa.Column('create_date', sa.DateTime(), nullable=False),
    sa.ForeignKeyConstraint(['question_id'], ['question.id'], ondelete='CASCADE'),
    sa.PrimaryKeyConstraint('id')
    )
    # ### end Alembic commands ###


def downgrade():
    # ### commands auto generated by Alembic - please adjust! ###
    op.drop_table('answer')
    op.drop_table('question')
    # ### end Alembic commands ###
```

Python 2 | Tab Width: 8 | Ln 18, Col 1 | INS

Create table (2. run upgrade)

- Run ‘flask db upgrade’ to reflect upgrade statement onto the database.

```
scott@scott-virtual-machine: ~/projects/webp
(webp) scott@scott-virtual-machine:~/projects/webp$ flask db upgrade
INFO  [alembic.runtime.migration] Context impl SQLiteImpl.
INFO  [alembic.runtime.migration] Will assume non-transactional DDL.
INFO  [alembic.runtime.migration] Running upgrade  -> a9dca3de4599, empty messag
e
(webp) scott@scott-virtual-machine:~/projects/webp$
```

Create table (3. check the results)

- Check for database file

```
scott@scott-virtual-machine: ~/projects/webp
(webp) scott@scott-virtual-machine:~/projects/webp$ ls -al
total 44
drwxrwxr-x 5 scott scott  4096 11월  7 02:23 .
drwxrwxr-x 4 scott scott  4096 11월  7 00:26 ..
drwxrwxr-x 3 scott scott  4096 11월  7 02:12 app
-rw-rw-r-- 1 scott scott   171 11월  3 03:05 config.py
drwxrwxr-x 4 scott scott  4096 11월  7 02:12 migrations
drwxrwxr-x 2 scott scott  4096 11월  7 02:12 __pycache__
-rw-r--r-- 1 scott scott 20480 11월  7 02:23 webp.db
(webp) scott@scott-virtual-machine:~/projects/webp$
```

- Inspect the table structure via DB Browser for SQLite