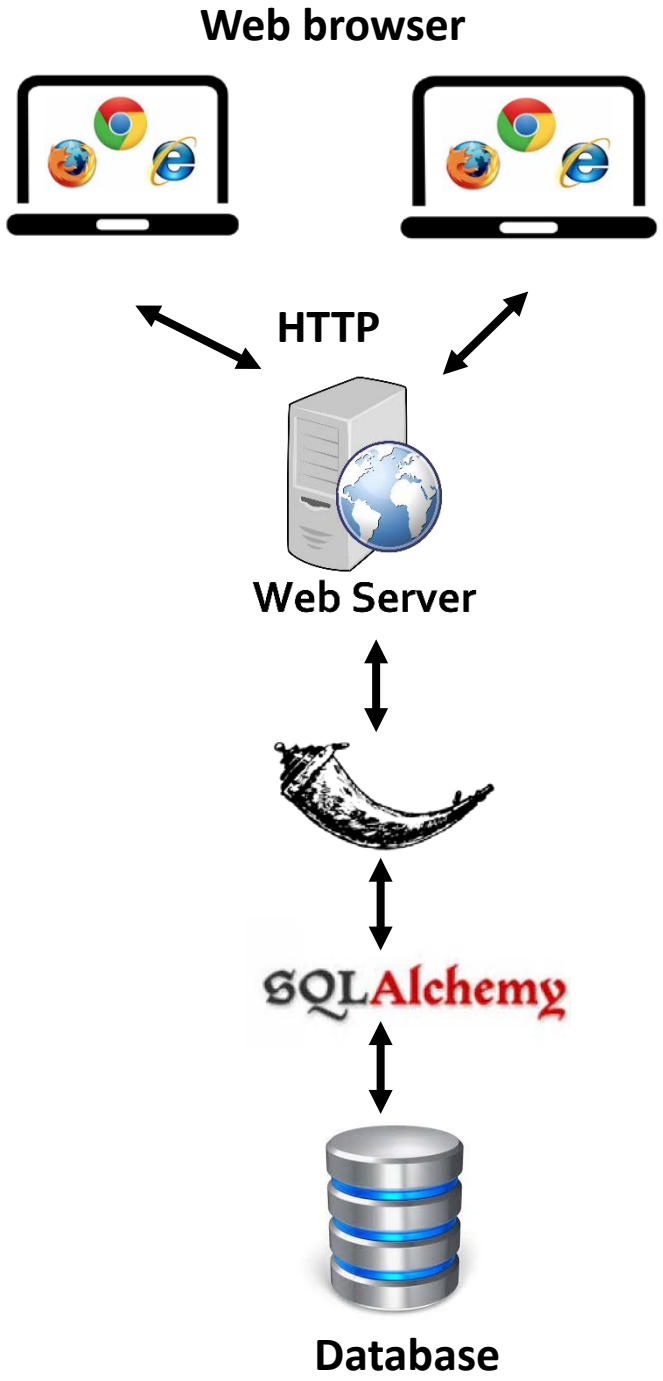
Persisting Data of an Application



Aims of this lecture

- Persist data in database
- Motivate Object Relational Mapping
- Introduce SQL Alchemy

Assessment 2 – Due Sept 15, 11:59 PM

- Create **static** HTML pages
  - Use Bootstrap for styling
  - Use of CSS separately is not necessary

- **Four** pages
  - Landing or Main Page
  - Item details page
  - Item creation page
  - Manage items page

- Support navigation using tags.
  - Questions on navigation

Topics covered Last Lecture

- Generating dynamic content in HTML
- Reusing HTML with templates
  - Template inheritance
- Introduction to Flask forms

Quiz

Topics covered Last Lecture

- Generating dynamic content in HTML

```
<div class="card-deck">
{% for hotel in hotels %}
<div class="card col-md-4">
<img class="card-img-top" src="{{'../static/img/' + hotel.image }}">

<div class="card-body">

<h4 class="card-title">{{hotel.name}}</h4>
<p class="card-text">{{hotel.description}}</p>

</div>
<div class="card-footer"><a href="#" class="btn btn-primary">Details</a></div>
</div>
{% endfor %}
</div>
```

Topics covered Last Lecture

- Reusing HTML with templates
  - Template inheritance

```
{% include "base.html" %}

{% block content %}
<h1>Sorry! this is not the page your are looking for.</h1>

{% endblock %}
```

Topics covered Last Lecture

- Introduction to Flask Forms

```
class ContactForm(FlaskForm):
        user_name = StringField('Name' )
        email = StringField('Email Address')
        submit = SubmitField("Submit")
```

```
{% import "bootstrap/wtf.html" as wtf %}
```

```
<div class="container">
<h2>Contact Form</h2>
<!--update the action and the method with relevant information--!>
{{wtf.quick_form(form, action=url_for('main.create_contact'))}}
</div>
```

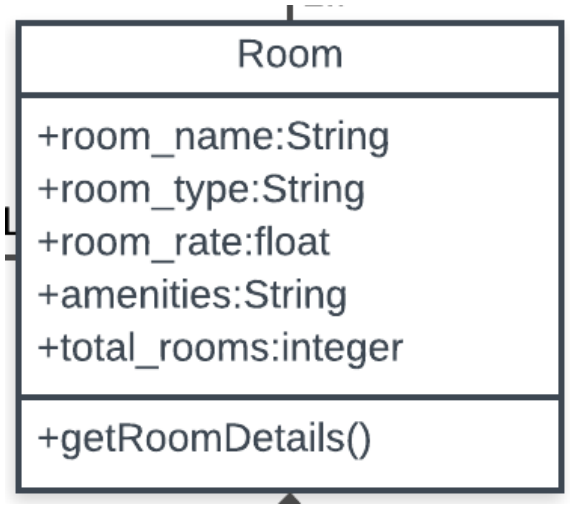
Model of the Web application

- Until recently, the most efficient way of storing data was in a relational database
  - Stores data in relations (tables)
  - Provides standard reliability - ACID
    - Atomicity – All or nothing is stored
    - Concurrency – Supports multiple requests
    - Isolation – Each transaction is isolated
    - Durability – Committed data is never lost

Relational Model (Formal)	Common terminology
Relation	Table
Tuple	Row
Attribute	Column
Domain	Column Type

The Relational Model

- Data should be represented as relations (tables).



room_table					
room_id	description	type	price	amenities	Num_rooms
0111	Standard room with two queen beds	Standard	110	free wifi	30
0112	something	King	120	free wifi, breakfast included	20
0118	something	Queen	110	free wifi	20
0119	Suitable for three adults	Triple	180	free wifi	10

Create an Object and Store it in the Database

- Define the attributes or member variables of the Class

week7_no_orm.zip

- Transform Object life cycle to SQL operations
  - Creating an object requires ‘insert’ operation on the database table
  - Updating an object requires ‘update’ operation on the database table
  - Retrieving details of an object requires ‘select’ operation on the database table
  - Deleting the object requires ‘delete’ operation on the database table

Code walkthrough

- Database connection
  - Reuse DB connection as much as possible
  - Open and close connection is time consuming
  - Having the connection open is not a good practice

week7_no_orm.zip

- Functions that transform objects to SQL statements

- The objects are added in the right sequence – parent object, child object

Questions?

Handling class relationships - Aggregation

- Adding the Parent Class (Hotel)
- Adding the ‘part-of’/’has-a’ Class (Room)
- Managing life cycle of both classes to maintain integrity
  - Insert
  - Select
  - Update
  - Delete

Handling class relationships - Inheritance

- The mapping can get more complex with these relationships

Impedance Mismatch

- An application deals with objects in the memory and needs to persist objects in relations.
  - Objects contain attributes which can be other objects (complex data structures)
  - Relations contains tuples or rows with name-value pairs.
  - The type of value in a tuple is standard data type – string, integer, float, binary

Translation is required from object model to relational model

Object Relational Mapping

- The most accepted solution today: **using middleware**.
  - A library that bridges OOP to RDBMS.
  - “Object/Relational Mapping” (ORM).

Characteristics of a ORM Solution

- **Transparent** (non-intrusive)
- **Efficient** (enables lazy-loading, selective updates, caching, etc.)
- **Powerful** (advanced querying capabilities, a lot of configuration options)
- **Supports OO idioms and concepts** (inheritance, polymorphism, etc.).

ORM with Flask

week7_orm.zip

- SQLAlchemy is an ORM for Python
  - A big advantage of using ORM is it is Database agnostic
  - Replace one database with another and it should ‘ideally’ work

- Flask uses SQLAlchemy to support databases

```
pip install Flask-SQLAlchemy
```

Flask SQL Alchemy supports different DBs

- A database is specified as a URL

Database	SQL Alchemy URL
MySQL	mysql://username:password@hostname/dbnae
Postgresql	postgresql://username:password@hostname/dbname
SQLLite	sqlite:///C:/pathtodatabase (windows) sqlite:////pathtodb (mac)

- Initialize SQLAlchemy
  - Every extension needs an initialization step

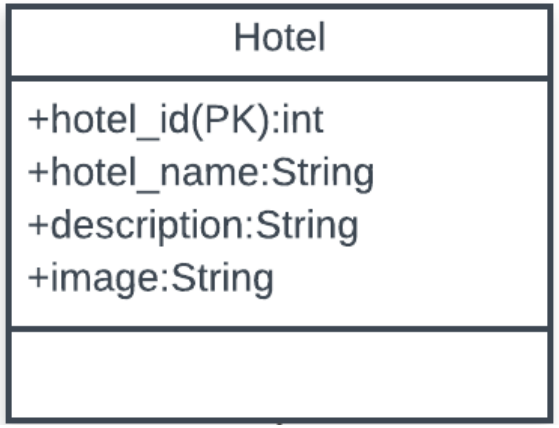
```
from flask_sqlalchemy import SQLAlchemy

db = SQLAlchemy()
```

```
app.config['SQLALCHEMY_DATABASE_URI']='sqlite:///hotel.sqlite'
db.init_app(app)
```

Creating a Class that Maps to a Table

- Create a class with base class `db.Model`
- List all the columns , their types and other column options




```
class Hotel(db.Model):

    __tablename__='hotels'
```

Good practice to specify table name

```
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(64), unique=True, index=True)
    description = db.Column(db.String(500), nullable=False)
    image = db.Column(db.String(60), nullable=False,default='default.jpg')
```

Column Types and Column Options

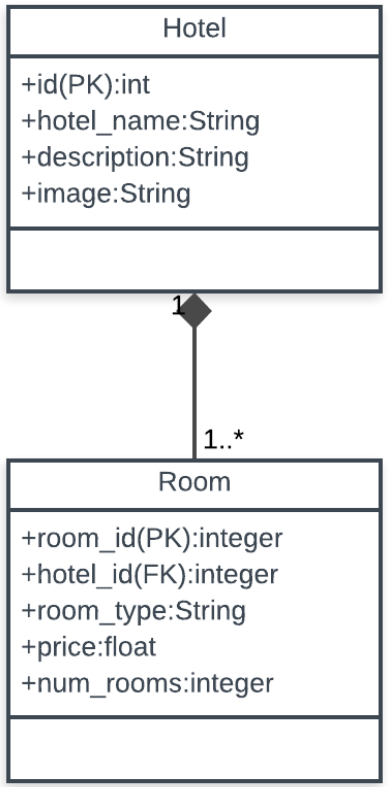
Column Type Name	Description
Integer	32 bit integer : int
String	Variable length string : str
Text	Variable length string optimized for large unbounded text : str
Date	DateTime.date
Time	DateTime.time
DateTime	DateTime.datetime
Numeric	Fixed point number : Decimal
Boolean	Boolean value

Options	Description
primary_key	Set to True, if column primary key
unique	True, duplicate values in the columns are not allowed – e.g. email id has to be unique in all the rows
index	True, if the column needs to be indexed, if you want any retrieve based on the column, it is good to set it to True.
nullable	True if this column can have empty values
default	A value that would be default. E.g if the column is not nullable, it would be good to assign a default value

Model one-to-many relationship

- Create two entities

- Define the Foreign Key in the table




```
class Room(db.Model):
        __tablename__ = 'rooms'

        id = db.Column(db.Integer, primary_key=True)
        room_type = db.Column(db.String(64), index=True)
        num_rooms = db.Column(db.Integer, nullable=False)
        description = db.Column(db.String(250), nullable=False)
         price = db.Column(db.Float, nullable=False)

        hotel_id = db.Column(db.Integer, db.ForeignKey('hotels.id'))
```

Questions?

Create a relationship in the class

- Relationship in the primary class (or multiplicity = 1)

- `backref='hotel'` indicates the attribute in Room class used to access Hotel

```
rooms = db.relationship('Room', backref='hotel')
```

Name of the class, having the foreign key
Not the table name

hotel is the name of the attribute to access in the Room class

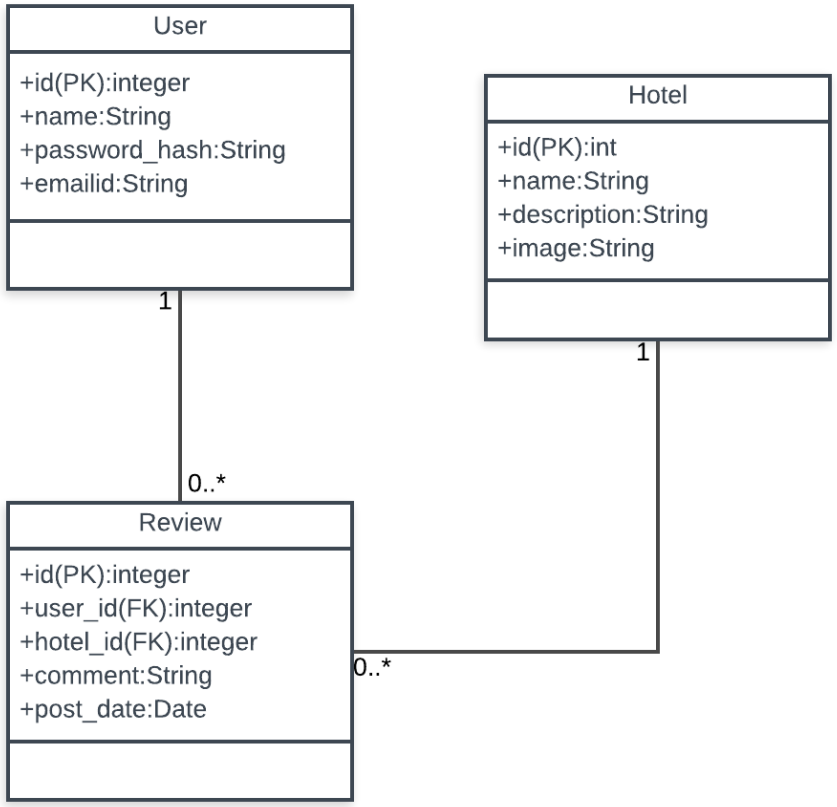
```
hotel1 = Hotel()
hotel1.rooms
```

```
standard1 = Room()
standard1.hotel
```

Model **one-to-one** relationship

```
room = db.relationship('Room', backref='hotel', uselist=False)
```

Model many-to-many relationship



- Hotel is reviewed by many users
- User reviews many Hotels
- Review – Association Class

Model many-to-many relationship

- Hotel is reviewed by many users
- User reviews many Hotels

```
class User(db.Model):
        __tablename__='users'
        id = db.Column(db.Integer, primary_key=True)
        name = db.Column(db.String(100), index=True, unique=True, nullable=False)
…
        comments = db.relationship('Review', backref='user')

        hotels = db.relationship('Hotel', secondary='reviews',
                                 backref=db.backref('commented_users'))
```

Table name of the Associative table

Object Hotel

Name of the attribute in the Hotel class

Questions?

Creating Objects

- Create object by passing named parameters

```
standard_room=Room(room_type='Standard', price=120, num_rooms=20,
                   description='Another standard room')
```

- Commit to the database

```
db.session.add(standard_room)
db.session.commit()
```

Querying Objects

- Run a select query on a model

Query filters – Returns a Query

Method	Description
filter_by()	Returns a query based on the filter parameter values
filter()	Returns a query with more flexible query parameter passing
order_by()	Returns the query that orders based on a criteria
group_by()	Returns a query that groups based on the criteria

Query Executors – Executes query

Method	Description
all()	Returns all the results of the query
first()	Returns the first result of the query

Querying Objects

querying_database.txt

Updating Objects

- Update attribute values of the object
- Commit to the database

```
#query the object - use db session to get the information in command line
standard_room=db.session.query(Room).filter_by(room_type='Standard').first()

print(standard_room.price)

#update the attribute value
standard_room.price=150

#commit the db session
db.session.commit()
```

View functions with SQLAlchemy

- Reading is a select query – query filter and query executor

Query the list of all hotels

```
@mainbp.route('/')
def index():
        tag_line='You need a vacation'
        hotels = Hotel.query.all() # query all the hotels
        form=ContactForm()
        return render_template('index_bootstrap.html', tag_line=tag_line,
                        form=form, hotels=hotels)
```

Query by id

```
@mainbp.route('/hotel/<hotelid>')
def get_hotel(hotelid):
        # use filter_by and get the first value
        hotel = Hotel.query.filter_by(id=hotelid).first()
        return('hotel_template.html', hotel=hotel)
```

View functions with SQLAlchemy

- Creating an Object

```
if form.validate_on_submit():

#create a new hotel with the information passed using FlaskForm.field.data
        new_hotel = Hotel(name=form.name.data,
        description=form.description.data,
                image=form.image.data)
# Using the SubForm form.room and its data to create the room object
        new_room = Room(room_type=.., .., hotel=new_hotel)
# add and commit to the database
        db.session.add(new_hotel)
        db.session.commit()
```

Questions?

Summary

- Persist data in database
  - Object and Relational data
- Motivate Object Relational Mapping
  - Impedance mismatch
- Introduce SQL Alchemy
  - Create, Read, Update, Delete

Thank you!