University of Glasgow

Entities, Relationships and Django models

Web Application Development 2

Entity-Relationship Model

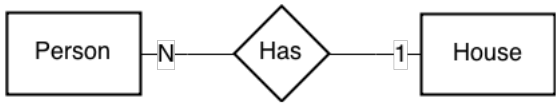
- Provides an abstract representation of the data and how they are related to each other
  - Developed by Peter Chen in 1976
- Lend themselves to being implemented in a database
- Three main components:
  - Entities
  - Relationships
  - Attributes

Notations

- Lots of different notations
  - Chen
  - Bachman
  - Barker
  - Martin
  - etc
- Each propose different ways to draw the ER model
  - We will be using a Compressed Chen Notation

Example ER Diagram

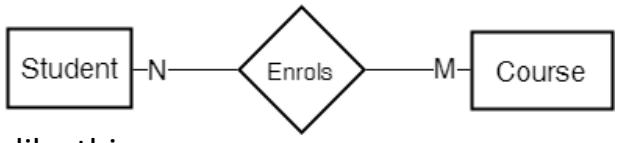
- Many people live in one house



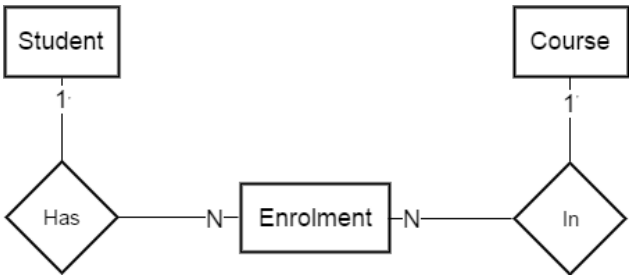
- **Rectangles** represent Entities
- **Diamonds** represent Relationships
- **1**,**N** or **M**, represents the cardinality of the relationship
  - N and M mean Many
  - Different notations use crows feet, etc.

Many to Many

- Many courses are taken by many students
  - Can be represented like this:

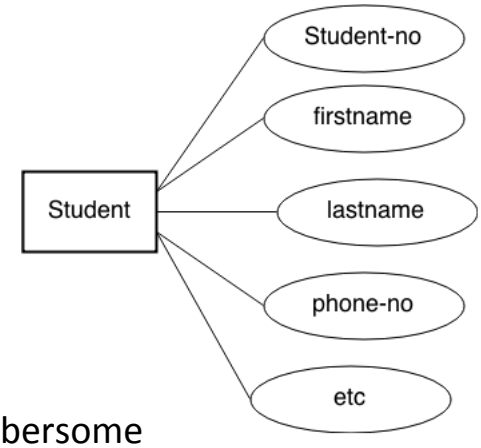


  - Or like this:



Chen vs Compressed Chen

- Chen Notation shows attributes as circles/ellipses



- This gets pretty cumbersome

Chen vs Compressed Chen

- So in compressed Chen, we only put entities and relationships in the diagram and then separately list the attributes

Field	Type
Student No	Char (8)
Firstname	Char (128)
SecondName	Char (128)
Phone-no	Char (15), formatted
etc	

- This way we can neatly represent the attributes and their types

Converting to Django Models

- In Django, every **model** is automatically assigned an **id**
- To create relationships between models, you refer to the **model** not the **id**
- e.g., given the House model then:

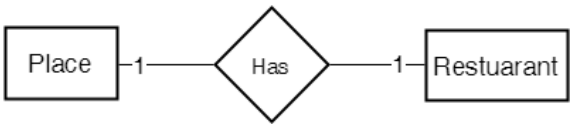
```
class Person(models.Model):
    house = models.ForeignKey(House)
```

denotes that many people live in one house

Example from Rango

```
class Category(models.Model):
   name = models.CharField(max_length=128, unique=True)
   views = models.IntegerField(default=0)
   likes = models.IntegerField(default=0)
   slug = models.SlugField(blank=True, unique=True)

class Page(models.Model):
   category = models.ForeignKey(Category)
   title = models.CharField(max_length=128)
   url = models.URLField()
   views = models.IntegerField(default=0)
```

Example one-to-one model



- A Place can optionally be a restaurant
- We start with the Place model

```
from django.db import models

class Place(models.Model):
   name = models.CharField(max_length=50)
   address = models.CharField(max_length=80)

   def __str__(self):
        return "%s the place" % self.name
```

Example one-to-one model (cont)

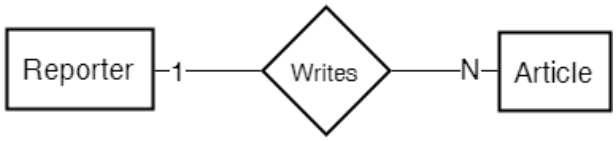
- The Restaurant model

```
class Restaurant(models.Model):
    place = models.OneToOneField(Place,
                on_delete=models.CASCADE, primary_key=True,
    )
    serves_hot_dogs = models.BooleanField(default=False)
    serves_pizza = models.BooleanField(default=False)

    def __str__(self):
        return "%s the restaurant" % self.place.name
```

- In Django, primary keys are normally auto-assigned integers, but here the primary key of a restaurant is its Place

https://docs.djangoproject.com/en/2.2/topics/db/examples/one_to_one/

Example many-to-one model



- A newspaper reporter may write many articles, but each article is written by just one reporter

```
from django.db import models

class Reporter(models.Model):
    first_name = models.CharField(max_length=30)
    last_name = models.CharField(max_length=30)
    email = models.EmailField()

    def __str__(self):
        return "%s %s" % (self.first_name, self.last_name)
```

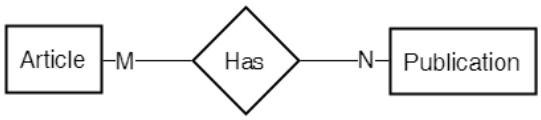
Example many-to-one model (cont)

```
class Article(models.Model):
   headline = models.CharField(max_length=100)
   pub_date = models.DateField()
   reporter = models.ForeignKey(Reporter,
                                on_delete=models.CASCADE)

   def __str__(self):
      return self.headline

   class Meta:
      ordering = ('headline',)
```

- Model metadata is optional “anything that’s not a field”, e.g. ordering options and human-readable singular / plural names

https://docs.djangoproject.com/en/2.2/topics/db/examples/many_to_one/

Example many-to-many model



- An article can be published in several places, and a publication may have several articles

```
from django.db import models

class Publication(models.Model):
   title = models.CharField(max_length=30)

   def __str__(self):
      return self.title

   class Meta:
      ordering = ('title',)
```

Example many-to-many model (cont)

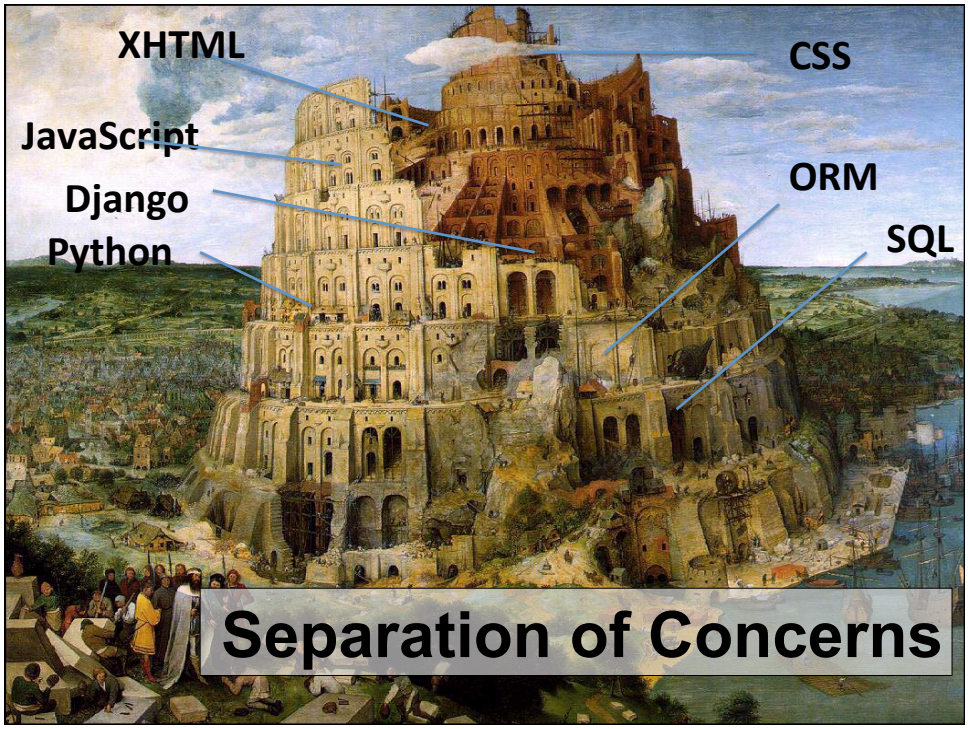
```
class Article(models.Model):
   headline = models.CharField(max_length=100)
   publications = models.ManyToManyField(Publication)

   def __ str __(self):
      return self.headline

   class Meta:
      ordering = ('headline',)
```

https://docs.djangoproject.com/en/2.2/topics/db/examples/many_to_many/

University of Glasgow

Cascading Style Sheets

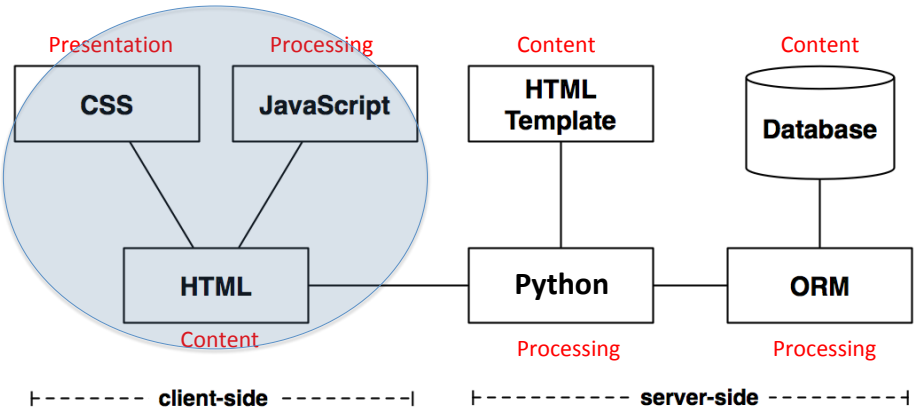
Web Application Development 2



Web Development Technologies

- Whilst developing web applications, the **minimum useful set of technologies** for a sufficiently complex application is **five:**
  - **Server-side Language (PHP, Ruby, Python, Java, etc)**
  - **Data Language (SQL)**
  - **Client-side Language (JavaScript)**
  - **Content Markup Language (XHTML)**
  - **Style Markup Language (CSS)**
- This turns out to be a headache when developing, especially when the languages mix together in one web page / file
- Maintenance becomes particulary difficult and codebase is especially fragile to change

Separation of Concerns



Separating the Presentation, Content and Processing on the client side

Style on the Web

- **Decisions of Style:** most aspects about any element of a web page can be controlled:
  - position, colour, size, font etc
- This can be achieved in any number of ways:
  - by using **Cascading Style Sheets** in combination with **XHTML**
  - by describing the page in XML and then using XSL to generate formatted XHTML
  - by using Cascading Style Sheets in combination with XML

Cascading Style Sheets

- Stylesheets describe the rendering of html elements
  - they specify stylistic aspects of **individual elements** or **all elements** of a particular kind
  - CSS consists of a set of **formatting rules,** which are specified in the following way:

```
selector {
    property1: value1;
    property2: value2;
    …
}
```

```
h3 {
    color: yellow;
    size: 18px;
    …
}
```

  - **selector** indicates the element (or set of), **property** refers to the stylistic aspect, and **value** is the specific configuration.

Find and Apply Pattern

```
p {
    font-size: 12pt;
    font-face: “Verdana”; }
```

Apply to all <p> elements

```
h1, h2, h3 {
    color: red;
    font-size: 18px; }
```

Apply to all <h1>, <h2>, <h3> elements

```
*{ text-align: left; }
```

Apply to all elements

```
#menu {
    padding: 45px 25px 0px 25px;
    border: none;
    height: 80px; }
```

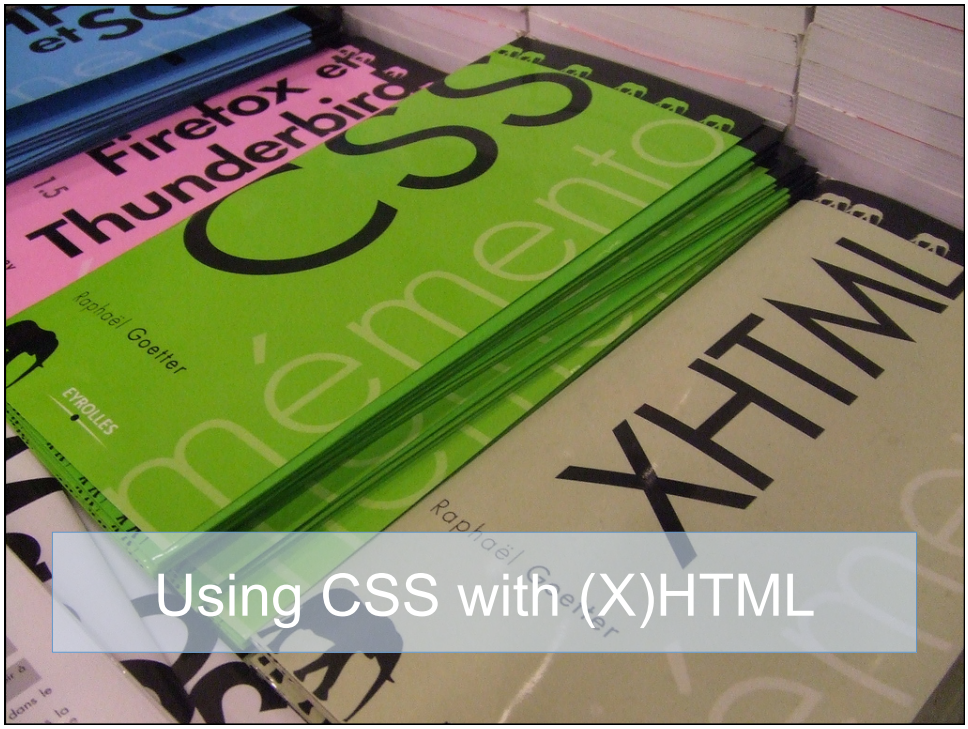
Apply to all elements with id=“menu”

Value and Units

- **Units** affect the colours, distances, and sizes of a whole host of properties of an element's style
- Numbers
  - can be integers or real numbers
- Percentages
  - real number followed by %
  - generally relative to some other number
  - e.g. font-size: 90% of the default or inherited value
- Colour
  - Name a colour (e.g. 'red'), functional rgb(255,0,0), or hexadecimal RGB codes (#FF0000). Property called **color**

Value and Units

- Length Units
  - inches (in), centimeters (cm), millimeters (mm), points (pt: 72pt = 1 inch), and picas (pc : 1pc = 12pt)
- Relative Length Units
  - em is relative to the given font-size value
  - e.g., font-size is 14pt, 1em = 14pt
  - ex is relative to the size of a lowercase x for the given font family
  - px is related to the size of a pixel on the device
    - px is generally the recommended unit to use



Using CSS with (X)HTML

Inline CSS Specification

- **Inline**: style information is added directly to one particular element using its **style** attribute
- CSS syntax is used with the **style** attribute in an HTML tag

```
<h3 style="color: yellow; font-size: 18px">
```

- This only affects this element, and others of the same type are not affected.
- Useful to **override** existing style, but **breaks** the **separation** of **content and presentation**

Embedded CSS Specification

- **Embedded**: Style rules can be specified in the <head> section of the document
- These rules will be applied to the entire document

```
<html>
    <head>
        <style>
            h3 {  color: yellow;  font-size: 18px;      }
        </style>
    </head>
    <body>
        … <h3> This text will appear yellow, 18px </h3> …
    </body>
</html>
```

External CSS Specification

- **External**: In a separate document which can be shared by several pages. The file extension is ".css"

```
<head>
    <link rel="stylesheet" href="master.css" type="text/css">
</head>
```

- This is generally the best method in terms of:
  - Separation of concerns
  - Maintenance
  - Performance