Django Summaries

Models

What is a model?

A model is a representation of the database schema we want. The easiest way to understand this is to compare a Django model with the direct SQL table we would create manually.

The link between SQL and Django's Model

If we wanted a table to store all of the items on our todo list, the table would look like this:

id	name	Due date
1	Something cool	2020-02-01
2	Another one	2020-03-05
3	And then this one	2020-04-09

Our model would reflect that database schema by looking like this:

```python
from django.db import models

class Todo(models.Model):
    name = models.CharField(max_length=100)
    due_date = models.DateField()
```

What's happening here is we are creating a model called "Todo" that inherits from Django's "Model" class. We are giving our class two properties; a name and a due date.

The `name` field is referencing a `CharField` which also comes from Django's `models` module. The `CharField` directly translates into a column in our todo table that only stores strings with a length up to 100 characters.

Likewise, the `due_date` field is referencing a `DateField` which directly translates into a column in our todo table that only stores date objects.

Fields

Django has many 'Fields' which provide us with different ways of representing the *type* of data we want in our models.

If we want to store an integer, we'd use an `IntegerField`. If we want to store a string, we could use a `CharField` or a `TextField`. If we want to store a boolean, we'd use an `BooleanField`. If we want to store a date or timestamp, we'd use a `DateField` or `DateTimeField`. etc.

Even though we are writing these data restrictions in Python, Django translates these fields into direct SQL restrictions on the data that can be stored in each column in our database. This helps make sure we store the correct type of data, and eliminates most of the basic errors we would make on our own.

Primary Keys and relationships

Primary keys are very important concepts when learning about databases and SQL. In a database we use primary keys as a way of linking two tables. Think of it as a way of specifying the relationship between two things.

Django lets us create relationships between our models that essentially are the same as creating relationships between database tables. We do this by using the `ForeignKey` model. We can also create relationships using the `OneToOneField` and `ManyToManyField`.

Meta

On a model we can specify a Python class called `Meta`. In this class we can specify more information about the model such as a custom database table name, plural representation of the model name and other information.

Model methods

It is common practice to add methods to a model. These methods are normally meant to perform a task on a specific instance. For example we could specify a method that returns a shorter version of the instance's title, such as below:

```python
from django.db import models

class Task(models.Model):
    title = models.CharField(max_length=100)

    def get_short_task_title():
        return title[:10]  # return the first 10 characters
```