Programação de Sistemas para Internet

Prof. Diego Cirilo

Aula 07: Models

Dados do sistema

- Como armazenar os dados do sistema?
- Conteúdo, dados de usuário, etc.
- SGBD - Sistema de Gerenciamento de Banco de Dados
- SQL - *Structured Query Language*

ORM

- Gerenciar comandos SQL pode ser complicado.
- ORM - *Object Relational Mapping*
- Podemos tratar os dados do BD como objetos

Models Django

- No Django o ORM é feito nos *Models* (`models.py`)
- Nele definimos os modelos de dados, seus atributos e comportamento.
- Cada modelo é uma classe Python que herda de `django.db.models.Model`
- Cada atributo de um model representa um campo no BD
- O Django se responsabiliza por gerenciar as tabelas, *queries*, etc.
- O Django também cria campos automaticamente, como ID.

Exemplo

```python
from django.db import models

class Pessoa(models.Model):
    nome = models.CharField(max_length=30)
    sobrenome = models.CharField(max_length=30)
```

```sql
CREATE TABLE myapp_pessoa (
    "id" bigint NOT NULL PRIMARY KEY GENERATED BY DEFAULT AS IDENTITY,
    "nome" varchar(30) NOT NULL,
    "sobrenome" varchar(30) NOT NULL
);
```

Alguns tipos de dados

- CharField: Textos curtos (nomes, títulos).
- TextField: Textos longos (descrições, artigos).
- IntegerField: Números inteiros (idades, quantidades).
- FloatField: Números decimais (preços, médias).
- BooleanField: Valores booleanos (Verdadeiro/Falso).
- DateField: Datas (aniversários, datas de criação).
- DateTimeField: Datas e horas (eventos, logs).
- Referência

Alguns tipos de dados

- EmailField: Endereços de e-mail (validação automática).
- ForeignKey: Relacionamentos um-para-muitos.
- ManyToManyField: Relacionamentos muitos-para-muitos.
- OneToOneField: Relacionamentos um-para-um.
- FileField: Arquivos.
- ImageField: Imagens.
- Referência

Algumas opções dos dados

- `max_length` - tamanho máximo para texto
- `null` - se vazio usa o `NULL` do SGBD
- `blank` - o campo pode ser vazio se `True`, por padrão é `False`
- `default` - valor padrão
- `unique` - se `True` o valor deve ser único na tabela
- `choices` - lista de valores possíveis

Relacionamentos

- Relacionam modelos
- Recebem como argumentos o nome da classe relacionada.
- ForeignKey: chave estrangeira
- ManyToManyField: Relacionamentos muitos-para-muitos.
- OneToOneField: Relacionamentos um-para-um.

Relacionamentos

- `on_delete` - define o que ocorre quando o objeto é removido
- `on_delete=models.CASCADE` - deleta os objetos relacionados junto
- `on_delete=models.SET_NULL` - escreve `NULL`
- Referência

Classe Meta

- Subclasse que permite algumas informações extras
- Ex.
  - `verbose_name`
  - `verbose_name_plural`
  - `ordering`

  -Referência

Acessando dados dos Models

- A *view* é responsável por acessar os dados
- É possível fazer *queries* através do objeto do Model
- `Model.objects.all()` - retorna *tudo*
- `Model.objects.filter()` - permite *filtrar* os dados
- `Model.objects.get(pk=4)` - seleciona o objeto específico
- Retornam `QuerySets`
- Os resultados podem ser enviados para o template no `context`

Filter

- Parecido com `WHERE` do SQL
- Padrão:
  - `campo__condicao=valor`
- Ex.
  - `Alunos.objects.filter(idade__lte=18)`

Filter Lookups

- `exact`
- `iexact`
- `contains`
- `startswith`
- `endswith`
- Referência

Configurações do BD

- No arquivo `settings.py` há a seção `DATABASES`
- Podemos configurar diversos SGBDs
- Ex. MySQL, PostGres, SQLite, etc.
- Para desenvolvimento o SQLite é simples e exige menos configuração.
- No sistema final devemos usar um SGBD mais completo (veremos em ICS).

Configurações de BD

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}
```

Migrations

- Arquivo com comandos para o SGBD
- Permite recriar a estrutura (*schema*) do BD em qualquer computador.
- Novas migrações devem ser criadas sempre que alteramos os dados em Models
- `python manage.py makemigrations`
- Para **aplicar** as *migrations*, ou seja, criar/alterar as tabelas no SGBD:
- `python manage.py migrate`
- É importante sempre lembrar de criar/aplicar as migrations.

Django Admin

- O Django foi pensado para facilitar o processo de desenvolvimento
- Um projeto Django já possui uma interface de administração pronta
- Para ativar, temos que:
  - Adicionar os models ao arquivo `admin.py`
  - Criar um *superuser* do sistema
  - Executar as *migrations*

Django Admin

```
from django.contrib import admin

from .models import Tarefa

admin.site.register(Tarefa)
```

Django Admin

- Para criar o *superuser*
- `python manage.py createsuperuser`
- Para criar as *migrations*
- `python manage.py makemigrations`
- Para executar as *migrations*
- `python manage.py migrate`

Django Admin

- Para configurar a língua do sistema (incluindo o *admin*):
  - Altere `LANGUAGE_CODE` no `settings.py` para `pt-br`
- Na listagem de tarefas aparece `Tarefas object(1)`, esse é o resultado do `print` em um objeto da classe `Tarefas`
- Para imprimir algo mais interessante, escrevemos o método `__str__` para a classe `Tarefas`

```
def __str__(self):
    return self.nome
```

Tarefa

- Crie um site de tarefas
- Cada tarefa deve ter: nome, status e prazo
- Marque as tarefas que estão atrasadas

Tarefa

- Crie um blog simples
- O blog deve ter um *header* com o título e um *footer* com informações do desenvolvedor
- O conteúdo do *blog* deve ser apenas uma imagem, um título, o texto e a data de publicação.
- Todas essas informações devem existir no BD
- Crie um *superuser* e cadastre as notícias pela página de *admin*

Tarefa

- Converta o site anterior dos atletas em um sistema;
- Crie os *models* para *Equipe*, *Atleta* e

Dúvidas?