Django: QuerySets e ORM

guilherme-lima

27/01/2020

**Django: QuerySets e ORM**

O mapeador de objeto-relacional [**ORM**](https://pt.wikipedia.org/wiki/Mapeamento_objeto-relacional) do **Django** é compatível com MySQL, PostgreSQL, SQLite e Oracle.

**O que o ORM do Django faz?**

O ORM do Django é fornecido com construções especiais de abstração que podem ser usadas para criar consultas complexas ao banco de dados. Em outras palavras, no lugar de realizar uma ação direta no banco de dados com código SQL por exemplo, utilizamos o ORM como ponte de comunicação entre o banco e a aplicação.

**O que é um QuerySet?**

**QuerySet** é um conjunto de ações que serão realizadas no banco de dados, ou seja, podemos criar, buscar, atualizar ou deletar os dados sem escrever a query SQL será executada no banco.

Abaixo, segue uma lista com as principais ações de um QuerySet:

**Ações de um QuerySet**

Usaremos como base o seguinte modelo, usado no curso [**Fundamentos de Django da Alura**](https://cursos.alura.com.br/course/fundamentos-django-2):

from django.db import models

class Pessoa(models.Model):

nome = models.CharField(max\_length=200)

email = models.CharField(max\_length=200)

def \_\_str\_\_(self):

return self.nome

Podemos também criar uma instância de um objeto na memória e posteriormente salvar no banco de dados usando o método save:

pessoa = Pessoa(nome='Adriélly', email='adrielly@alura.com')

pessoa.save()

A ação anterior, executa o **insert no banco de dados.** Vimos como criar um objeto na memória primeiro e depois persistir no banco de dados, mas também podemos criar o objeto e persistir no banco de dados em uma única operação usando o método create:

Pessoa.objects.create(nome='Guilherme', email='gui@alura.com')

Já para buscar todas as pessoas cadastradas no banco, usamos o método all:

Pessoa.objects.all()

Caso queira buscar uma instância do banco de dados que contenha o nome *Guilherme* por exemplo, posso utilizar o método get:

pessoa = Pessoa.objects.get(nome='Guilherme')

O método get permite recuperar um **único objeto do banco de dados** (por exemplo, uma primary key). Se nenhum resultado for retornado pelo banco de dados, esse método gerará uma exceção de **DoesNotExist** indicando que o objeto não existe e, se o banco de dados retornar mais de um resultado, gerará uma exceção **MultipleObjectsReturned**, indicando que existe múltiplos objetos retornados.

Para atualizar o email do Guilherme por exemplo, utilizamos também o método save, porém, dessa vez, **o método executa uma instrução SQL UPDATE**, já que temos uma instância.

pessoa.email = 'guilherme@alura.com'

pessoa.save()

As alterações feitas no objeto não são persistidas no banco de dados até que o método save seja executado.

Você pode excluir determinados resultados do seu QuerySet usando o método delete:

pessoa = Pessoa.objects.get(id=1)

pessoa.delete()

**Filtros**

Para os próximos exemplos, considere o seguinte modelo:

from django.db import models

from datetime import datetime

class Receita(models.Model):

nome\_receita = models.CharField(max\_length=200)

modo\_preparo = models.TextField()

date\_receita = models.DateTimeField(default=datetime.now, blank=True)

publicada = models.BooleanField(default=False)

def \_\_str\_\_(self):

return self.nome\_receita

Para filtrar um QuerySet, você pode usar o método filter. Por exemplo, podemos recuperar todas as receitas publicadas, como mostra o código abaixo:

receitas\_publicadas = Receita.objects.filter(publicada=True)

É possível também filtrar por vários campos:

receita\_publicada\_chocolate = Receita.objects.filter(publicada=True, nome\_receita='Bolo de chocolate')

**Ordenação**

Também é possível recuperar todos os objetos ordenados com o método order\_by, por exemplo, todas as receitas ordenadas pelo nome:

receitas\_ordenadas = Receita.objects.order\_by('nome\_receita')

A ordem crescente está **implícita**. Você pode indicar a ordem decrescente com um prefixo de sinal negativo, assim:

receitas\_ordenadas = Receita.objects.order\_by('-nome\_receita')

**O que aprendemos neste artigo**

Ao invés de escrever o código SQL, podemos pedir de uma forma elegante para o ORM do Django realizar essa ação por nós. Você também pode concatenar quantos filtros desejar para um QuerySet e não atingirá o banco de dados até que o QuerySet seja avaliado. Todos os exemplos mostrados acima, foram executados nos [**cursos de Django da Alura**](https://www.alura.com.br/curso-online-integracao-modelos-django-2).

[Cursos de Programação](https://www.alura.com.br/cursos-online-programacao)