Работа с базой данных.

- Документация: https://docs.djangoproject.com/en/2.0/topics/db/

Шередеко Василий, piphon@gmail.com

Django ORM

ORM - Object Relational Mapping

> Технология программирования, которая связывает базы данных с концепциями ООП
> **@wikipedia**

Django ORM

Возможности:

- Декларативное описание таблиц (`models.Model`)
- Хранение изменений структур таблиц
- Запросы к таблицам
- Скорость разработки
- Абстракция от SQL

`django.db.models.BooleanField`

Поле для хранения логических значений: `True` и `False`

`django.db.models.SmallIntegerField`

Поле для хранения целых чисел в промежутке от `-32768` to `32767`.

`django.db.models.IntegerField`

Поле для хранения целых чисел в промежутке от `-2147483648` to `2147483647`.

`django.db.models.BigIntegerField`

Поле для хранения целых чисел в промежутке от `-9223372036854775808` до `9223372036854775807`.

`django.db.models.PositiveSmallIntegerField`

Поле для хранения целых чисел в промежутке от `0` to `32767`.

`django.db.models.PositiveIntegerField`

Поле для хранения целых чисел в промежутке от `0` to `2147483647`.

`django.db.models.CharField`

Поле для хранения строк

Параметры:

- `max_length` - максимальная длина значения

`django.db.models.FloatField`

Поле для хранения чисел с плавающей точкой

`django.db.models.DecimalField`

Поле для хранения чисел с фиксированной точкой

Параметры

- `max_digits` - максимальное количество цифр.
- `decimal_places` - количество цифр после точки.

`django.db.models.DateField`

Поле для хранения дат

`django.db.models.TimeField`

Поле для хранения времени

`django.db.models.DateTimeField`

Поле для хранения даты и времени

Параметры

- `auto_now_add` - использовать текущие дату и время как значение по умолчанию.

`django.db.models.EmailField`

Поле для хранения email

`django.db.models.FileField`

Поле для хранения файлов

Параметры

- `upload_to` - путь где хранятся файлы или функция:

```
def upload_to(instance, filename): ...
```

`django.db.models.ImageField`

Поле для хранения файлов с изображениями

Параметры

- `upload_to` - путь где хранятся файлы или функция:

```
def upload_to(instance, filename): ...
```

`django.db.models.ForeignKey`

Поле для хранения ссылки на внешнюю модель (один-ко-многим).

Параметры

- `related_name` - название обратного аттрибута.
- `on_delete` - поведение при удалении

Дополнительно

https://en.wikipedia.org/wiki/One-to-many_(data_model)

`django.db.models.OneToOneField`

Поле для хранения ссылки на внешнюю модель

Параметры

- `related_name` - название обратного аттрибута.
- - `on_delete` - поведение при удалении

Дополнительно

https://en.wikipedia.org/wiki/Many-to-many_(data_model)

Значения `on_delete`:

- `models.CASCADE` - каскадное удаление
- `models.PROTECT` - запрещает удаление
- `models.SET_NULL` - выставляет значение поля в None
- `models.SET_DEFAULT` - выставляет значение по умолчанию
- `models.SET` - выставляем
- `models.DO_NOTHING` - ничего не делаем

`django.db.models.ManyToManyField`

Поле для хранения ссылки на внешнюю модель

Параметры

- `related_name` - название обратного аттрибута.

Дополнительно

https://en.wikipedia.org/wiki/Many-to-many_(data_model)

Общие параметры

- `default` - значение по умолчанию.
- `null` - значение поля может отсуствовать.
- `blank` - значение поля может быть пустым.
- `editable` - значение можно редактировать.
- `unique` - есть уникальный индекс по полю.

Миграции

Миграции предназначены для переноса изменений из моделей в базу данных.

- Документация: https://docs.djangoproject.com/en/2.0/topics/migrations/

Миграции: создание новой миграции на основе изменений в моделях.

```
python manage.py makemigrations <app>
```

Миграции: применение миграций

```
python manage.py migrate <app>
```

Миграции: применение всех миграций

```
python manage.py migrate <app>
```

Миграции: список миграций

```
python manage.py showmigrations
```

Запросы.

- Документация: https://docs.djangoproject.com/en/2.0/ref/models/querysets/

Запросы

`create(**kwargs)`

Создание нового объекта в БД

```
obj = Article.objects.create(name='Статья')
```

Запросы

`get(**kwargs)`

Получить один объект из запроса

```
obj = Article.objects.get(name='Hello world!')
```

Ошибки:

- `DoesNotExist` - если результат пустой
- `MultipleObjectsReturned` - если в результате найдено более одного объекта.

Запросы

`get_or_create(*, defaults, **kwargs)`

Получить объект из БД или создать его, если он там отсуствует

```
obj, created = Article.objects.create(name='Статья')
```

Ошибки:

- `MultipleObjectsReturned` - если в результате найдено более одного объекта.

Запросы

`delete`

Удалить объекты из результат запроса в БД

```
queryset = Article.objects.all().delete()
```

Запросы

`all`

Получить копию запроса

```
queryset = Article.objects.all()
```

Запросы

`filter(**kwargs)`

Отфильтровать объекты из результата запроса

```
queryset = Article.objects.filter(name='FIFA 2018')
```

Запросы

`exclude(**kwargs)`

Убрать объекты из результата запроса

```
queryset = Article.objects.exclude(name='FIFA 2018')
```

Запросы

`order_by(*fields)`

Отсортировать объекты в результате запроса

```
queryset = Article.objects.order_by('name')
```

Запросы

`distinct`

Убрать одинаковые объекты из результата запроса

```
queryset = Article.objects.distinct()
```

Запросы

`count`

Получить количество объектов в результате запроса

```
length = Article.objects.count()
```

Запросы

`values(*fields)`

Получить результат в виде коллекции словарей

```
items = Article.objects.values(`id`, `name`)
```

`values_list(*fields)`

Получить результат в виде коллекции кортежей

```
items = Article.objects.values_list(`id`, `name`)
```

Запросы

`select_related`

Получить связанные экземпляры моделей в том же запросе.

```
queryset = Article.objects.select_related(`directory`)
```

`prefetch_related`

Получить связанные экземпляры моделей в отдельно запросе.

```
queryset = Article.objects.prefetch_related(`directory`)
```

Запросы

`<field>__exact`

`<field>__iexact`

Поиск по точному значению.

```
queryset = Article.objects.filter(id__exact=10)
```

```
queryset = Article.objects.filter(id=10)
```

Запросы

`<field>__contains`

`<field>__icontains`

Поиск по части значения

```
queryset = Article.objects.filter(name__contains="elo")
```

Запросы

`<field>__in`

Поиск по вхождению в коллекцию или другой запрос.

```
queryset = Article.objects.filter(id__in=[1, 2, 3])
```

```
other_qs = Article.objects.filter(...)
queryset = Article.objects.filter(id__in=other_qs)
```

Запросы

`<field>__gt` (`>`)

`<field>__gte` (`>=`)

`<field>__lt` (`<`)

`<field>__lte` (`<=`)

Сравнение со значением.

```
queryset = Article.objects.filter(id__gt=10)
```

Запросы

`<field>__isnull`

Проверка на присутствие значения.

- `True` - значения нет (`NULL`)
- `False` - есть значение

```
queryset = Article.objects.filter(parent__isnull=False)
```