Модели + ORM = данные

Краткая теория баз данных. Введение в django-ORM. Подключение и создание базы данных. Несколько слов о миграциях. Работа с данными через консоль. Дополняем демо-сайт "About me"

[Краткая теория баз данных(БД)](#_c713xnuw3xem)

[Введение в Django ORM](#_qgye5r7e5sog)

[Подключение БД](#_nf8xqspq9y8a)

[Выбор БД](#_bagr4g6fvkps)

[Установка и настройка](#_n9kaqrm7t2tx)

[Работа через консоль](#_4rk73dktvr8u)

[Добавим супер-пользователя](#_2txoknx9vdp5)

[Запускаем интерактивную консоль django](#_e7zpxm2nd5un)

[Сохраняем его в БД:](#_tnenhzf0nqen)

[Несколько слов о миграциях](#_b6jjfgmvrksv)

[Создаем собственную модель](#_hg2all4hcff9)

[Определяем поля](#_dcmgmacqnip)

[Определяем типы полей](#_a0levpxip9ru)

[Создаем модель](#_j4e0mq9dvffb)

[Создадим таблицу в БД](#_7505r6hnwqnc)

[Создаем объекты и сохраняем в БД](#_kpttjfr3tsig)

[Отображаем данные из базы на html-странице](#_c79xrfx887gw)

[Домашнее задание](#_owpsqd9opgcf)

[Дополнительные материалы](#_y937sk8fclye)

[Используемая литература](#_uvp6qax5r1ok)

# 

# Краткая теория баз данных(БД)

База данных - набор сведений, хранящихся некоторым упорядоченным способом. Можно сравнить базу данных со шкафом, в котором хранятся документы. Иными словами, база данных - это хранилище данных. Сами по себе базы данных не представляли бы интереса, если бы не было систем управления базами данных (СУБД).

Система управления базами данных - это совокупность языковых и программных средств, которая осуществляет доступ к данным, позволяет их создавать, менять и удалять, обеспечивает безопасность данных и т.д. В общем СУБД - это система, позволяющая создавать базы данных и манипулировать сведениями из них. А осуществляет этот доступ к данным СУБД посредством специального языка - SQL.

SQL - язык структурированных запросов, основной задачей которого является предоставление простого способа считывания и записи информации в базу данных.

Создавая базу данных, мы стремимся упорядочить информацию по различным признакам для того, чтобы потом извлекать из нее необходимые нам данные в любом сочетании. Сделать это возможно, только если данные структурированы. Структурирование - это набор соглашений о способах представления данных. Понятно, что структурировать информацию можно по-разному. В зависимости от структуры различают иерархическую, сетевую, реляционную, объектно-ориентированную и гибридную модели баз данных. Самой популярной на сегодняшний день является реляционная структура.

Если говорить упрощенно, базы данных представляют собой связанные таблицы. Каждая из таблиц содержит информацию об объектах одного типа. Из названий таблиц становиться понятно, что данные в каждой таблице принадлежат одной и той же группе объектов. Каждая строка в этих таблицах однозначно определяет один объект из соответствующей группы. Вообще, база данных может состоять из одной или нескольких таблиц. Запись, в свою очередь, состоит из нескольких полей, каждое из которых содержит элемент данных об объекте.



рис. пример таблицы “Сотрудники”

Современные базы данных - очень мощные инструменты, позволяющие осуществлять очень гибкую выборку данных, хранить и обрабатывать огромные массивы, следящие за целостностью.

# Введение в Django ORM

Django поддерживает четыре системы управления базами данных:

PostgreSQL (<http://www.postgresql.org> ).

SQLite 3 ([http://www.sqlite.org](http://www.sqlite.org/)).

MySQL ([http://www.mysql.com](http://www.mysql.com/)).

Oracle (<http://www.oracle.com> ).

По большей части движки для перечисленных баз данных хорошо работают с ядром среды Django.

Django предоставляет абстрактный уровень (“модели”) для создания структуры и управления данными вашего приложения, что позволяет абстрагироваться от особенностей реализации конкретной БД и писать запросы на более высосом (объектном) уровне. Знание SQL не будет лишним, но и без его знания можно успешно писать довольно сложные и эффективные запросы.

ORM (англ. Object-Relational Mapping, рус. объектно-реляционное отображение) — технология программирования, которая связывает базы данных с концепциями объектно-ориентированных языков программирования, создавая «виртуальную объектную базу данных».

Модели отображают информацию о данных, с которыми вы работаете. Они содержат поля и поведение ваших данных. Обычно одна модель представляет одну таблицу в базе данных.

Вот пример модели, которая определяет гипотетического человека(Person), с именем(first\_name) и фамилией(last\_name):

|  |
| --- |
| from django.db import models  class Person(models.Model):  first\_name = models.CharField(max\_length=30)  last\_name = models.CharField(max\_length=30) |

first\_name и last\_name fields\_ модели. Каждое поле определено как атрибут класса, и каждый атрибут соответствует полю таблицы в базе данных.

Самая важная часть модели – и единственная обязательная – это список полей таблицы базы данных которые она представляет. Поля определены атрибутами класса. Тип поля показывает, в каком виде ваши данные будут храниться в базе, и что важнее, в качестве каких типов данных будут доступные при обращении к ним через атрибуты объекта. Так же тип поля определяет валидацию данных, перед сохранением в БД.

Валидация - проверка данных на корректность.

Еще раз, модель является посредником между данными в БД и данными в виде python-объекта. Вы создаете объект, его атрибутам присваиваете значения, а при сохранении в БД, модель транслирует ваш python-объект в SQL запрос. И наоборот, при чтении данных, модель формирует SQL запрос и преобразует его в python-oбъект/список python-объектов.

Кажется сложным? Пара примеров и всё станет на свои места.

# Подключение БД

Прежде, чем начать пользоваться базой, её нужно подключить.

## Выбор БД

Несмотря на то, что sqlight3 является наиболее медленной базой, и практически не используется в высоконагруженных проектах, мы будем использовать именно её. Причина проста - её вообще не нужно ни устанавливать, ни настраивать, она по умолчанию доступна в любой ОС. При необходимости, вы всегда можете перейти к любой БД, прочитав способ подключения. И помните, переход к другой БД совершенно не затронет код работы с базами.

Еще одном важным преимуществом(для начинающих django-программистов) является то, что sqlight3 представляет собой просто файл(с именем db.sqlite3 по умолчанию), и если что-то напортачите ( а в начале вы часто будете ошибаться) можно просто удалить данный файл и попробовать заново.

## Установка и настройка

В Django 1.8 sqlight3 подключена по умолчанию, проверить это можно, зайдя в файл settings.py и увидев строки

|  |
| --- |
| DATABASES = {  'default': {  'ENGINE': 'django.db.backends.sqlite3',  'NAME': os.path.join(BASE\_DIR, 'db.sqlite3'),  }  } |

или поменяв на таковые, если в вашей версии значения проекта по умолчанию другие

Чтобы создать файл с пустой базой данных, выполните команду для автоматического создания таблиц базы данных:

|  |
| --- |
| python manage.py migrate |

При успешном выполнении появится файл db.sqlite3 в корне проекта.

## Работа через консоль

Важно научиться работать с ORM через консоль. Почему? Во-первых, так вы быстро сможете добавлять данные в базу и читать из базы, без необходимости создавать формы и вьюшки (функции-представления). Во-вторых, вы полностью будете видеть весь процесс работы с ORM. И, в-третьих, рано или поздно вам все равно придется этому научиться.

Приступим.

Самый быстрый и простой способ добавить данные в базу, это создать супер-пользователя (администратора). Модель для пользователей уже существует в недрах Django. Подробно про эту модель и его поля [здесь](http://djbook.ru/rel1.8/ref/contrib/auth.html#django.contrib.auth.models.User).

### Добавим супер-пользователя

|  |
| --- |
| python manage.py createsuperuser |

Далее укажите логин, е-мэйл и пароль (если забудете пароль, не беда, его всегда можно изменить).

### Запускаем интерактивную консоль django

|  |
| --- |
| python manage.py shell |

Чтобы использовать модель User, импортируем её:

|  |
| --- |
| from django.contrib.auth.models import User |

Выполняем самый первый запрос:

|  |
| --- |
| User.objects.all() |

Получаем:

|  |
| --- |
| >> [<User: booblegum>] |

Создаем еще одного пользователя:

|  |
| --- |
| user = User.objects.create\_user('john', 'lennon@thebeatles.com', 'johnpassword') |

### Сохраняем его в БД:

|  |
| --- |
| user.save() |

Получаем всех пользователей в базе:

|  |
| --- |
| User.objects.all() |

Получаем:

|  |
| --- |
| >> [<User: booblegum>, <User: john>] |

Разбираем подробнее получившееся:

Рассмотрим запрос:

|  |
| --- |
| User.objects.all() |

User - это модель. objects - менеджер объекта. Каждая модель содержит минимум один менеджер и называется он objects. Метод .all() менеджера ищет в базе все записи на класс User и возвращает их в виде списка python-объектов. Подробнее [здесь](http://djbook.ru/rel1.7/topics/db/queries.html#making-queries).

Поэкспериментируйте со списком полученных объектов, например, у User’ов есть атрибуты email и username. Вы уже знаете как получить значение атрибута?

Теперь рассмотрим создание пользователя:

Чтобы создать запись в БД, нужно:

1. Создать модель для данного объекта (у нас уже есть модель User).
2. Создать экземпляр класса модели и заполнить обязательные поля (это мы выполнили командой user = User.objects.create\_user('john', 'lennon@thebeatles.com', 'johnpassword')).
3. У экземпляра класса вызвать метод save().

# Несколько слов о миграциях

Предже, чем приступить к созданию собственных моделей, нужно немного сказать о миграциях, т.к. с версии django 1.7 они стали неотъемлемой частью фрэймворка.

Django использует миграции для переноса изменений в моделях (добавление поля, удаление модели и т.д.) на структуру базы данных. Миграции создавались в основном для автоматической работы, но вам необходимо знать, когда их создавать, запускать и как решать различные проблемы.

До версии 1.7, Django позволял только добавлять новые модели в базу данных; не было возможности изменять или удалять существующие модели, используя команду syncdb (предок команды migrate).

Сторонние инструменты, самый известный South, позволяли создавать и выполнять миграции. Со временем было решено перенести этот функционал в Django.

Django предоставляет три команды для работы с миграциями и структурой базы данных:

* migrate, которая отвечает за применение миграций, за откат миграций и за вывод статуса миграций.
* makemigrations, которая отвечает за создание новых миграций на основе изменений в моделях.
* sqlmigrate, которая выводит SQL запросы для миграции.

Стоит отметить, что миграции создаются и работают в контексте отдельного приложения. В частности, можно создать приложение, которое не использует миграции - такие приложения имитируют старое поведение и просто создают новые модели.

Следует рассматривать миграции, как систему контроля версий для базы данных. makemigrations отвечает за сохранение состояния моделей в файле миграции - аналог коммита - а migrate отвечает за их применение к базе данных.

Файлы с миграциями находятся в каталоге “migrations” приложения. Они являются частью приложения и должны распространятся вместе с остальным кодом приложения. Они должны создаваться при разработке, и потом применятся на машинах коллег, тестовом и “боевом” серверах.

Миграции работают идентично на одном и том же наборе данных. Это означает, что на сервере разработки, тестовом и “боевом” серверах вы получите один и тот же результат при одинаковых условиях выполнения миграций.

Django создаст миграции при любых изменениях модели или полей, даже тех параметров, которые не влияют на базу данных, т.к. единственный способ восстановить состояние моделей - это хранить все изменения в истории. Вам могут понадобиться эти параметры в миграциях данных в будущем (например, если вы добавите собственную проверку данных).

Подробнее [здесь](http://djbook.ru/rel1.8/topics/migrations.html).

# Создаем собственную модель

В рамках работы над нашим мини-проектом About ME, создадим модель места работы.

Определим поля места работы (возьмем за отправную точку список полей на hh.ru).

## Определяем поля

* Организация (в которой работали);
* Регион (нахождения организации);
* Сайт;
* Должность;
* Обязанности функции;
* Время работы (полных месяцев);

На данный момент полей достаточно.

Далее определяем типы полей, которые в свою очередь определят проверку и типы хранимых данных.

[Здесь](http://djbook.ru/rel1.7/ref/models/fields.html#field-types) вы найдёте полное описание всех типов полей. Если вы впервые работаете с моделями, рекомендуется изучить все доступные типы и подбирать наиболее подходящие.

## Определяем типы полей

* Организация (CharField - название организации будем хранить в виде обычного текста)
* Регион (CharField)
* Сайт (CharField)
* Должность(CharField)
* Обязанности функции (TextField - описание будет в произвольном виде, этот тип наиболее подходит)
* Время работы (PositiveIntegerField - целое значение, с дополнительной защитой от отрицательных значений)

## Создаем модель

models.py

|  |
| --- |
| from django.db import models  class Work(models.Model):  organization = models.CharField(verbose\_name='Организация', max\_length=32)  region = models.CharField(verbose\_name='Регион', max\_length=32, blank=True)  site = models.CharField(verbose\_name='Сайт', max\_length=64, blank=True)  position = models.CharField(verbose\_name='Должность', max\_length=16)  duties = models.TextField(verbose\_name='Обязанности')  period = models.PositiveIntegerField(verbose\_name='Время работы', default=1) |

**Обратите внимание:** поля типа CharField имеют параметры поля max\_legth - ограничивающие максимальное количество символов, это значение используется для валидации (проверки) данных перед сохранением в БД. Поля, имеющие параметр blank=True, являются необязательными при заполнении формы, если поле не заполнено, в базу сохраняется пустая строка. verbose\_name - отображаемое имя поля, будет использоваться при использовании форм. default - значение по умолчанию. Полный список параметров полей см. [здесь](http://djbook.ru/rel1.7/ref/models/fields.html#module-django.db.models.fields).

## 

## 

## Создадим таблицу в БД

Объекты (в данном случае наши “Работы”) будут сохраняться в БД, для этого им нужна таблица, построенная на основании нашей модели, для этого:

1. Добавить наше приложение в settings.py

settings.py (добавить)

|  |
| --- |
| INSTALLED\_APPS = (  'django.contrib.admin',  'django.contrib.auth',  'django.contrib.contenttypes',  'django.contrib.sessions',  'django.contrib.messages',  'django.contrib.staticfiles',  'mainapp'  ) |

Это необходимо, чтобы миграции знали, где искать models.py

1. Создать миграцию:

|  |
| --- |
| manage.py makemigrations |

В ответ получите нечто следующее:

|  |
| --- |
| >> Migrations for 'mainapp':  0001\_initial.py:  - Create model Work |

1. Применить миграцию (добавит таблицу в БД)

|  |
| --- |
| python3 manage.py migrate mainapp |

## 

## 

## 

## Создаем объекты и сохраняем в БД

1. Создаем объект “Работа”, через консоль

|  |
| --- |
| from mainapp.models import Work work = Work() work.organization = 'Test org' work.site = 'test.ru' work.duties = 'Тестовое описание...' work.save() |

1. Получаем список работ из базы, проверяем значения атрибутов:

|  |
| --- |
| works = Work.objects.all() works[0].organization works[0].duties works[0].period |

**Мини-задание:** Самостоятельно создайте несколько объектов “Работа”, попробуйте различные вариант задания значений для полей, поэкспериментируйте с результатами.

# Отображаем данные из базы на html-странице

Теперь наши данные хранятся в полноценной БД. Используя те же команды, что мы использовали в консоли, мы можем получить нужные нам данные во views.py и, используя шаблон, отобразить на нашей странице.

views.py

|  |
| --- |
| from .models import Work  def works(request):  place\_of\_work = Work.objects.all()  return render\_to\_response("works.html", {'place\_of\_work':  [work.organization for work in place\_of\_work]}) |

Подумайте и скажите, чем плоха текущая реализация?

Думайте… (ответ ниже)

Ответ: Мы снова обрабатываем отображение в функции-представлении, гораздо правильнее эту задачу поручить шаблону, передав ему полные объекты. Но это уже ваша задача.

# Домашнее задание

Продолжаем дорабатывать проект About ME, использую новые знания, полученные на текущем вебинаре, а также пользуясь документацией по django (все ссылки в “Дополнительных материалах”)

1. Пользуясь консольными командами, добавить свои места работы в БД.
2. По аналогии сделать модели для ваших хобби и мест учебы. Не забывайте выполнять миграции, после изменения/добавления моделей. Если всё сломается, не бойтесь, удаляйте файл БД и начинайте заново, эту цепочку полезно пройти несколько раз. Если нет желания вводить одни и те же данные много раз - подумайте, как можно этого избежать (вы программисты, ваша задача - избегать рутинных и монотонных задач). Используя данные из заполненной БД, отобразить всю информацию.  
   P.S. : Если не можете придумать поля для хобби, возьмите, например, такие: название, ссылка на иконку (путь к файлу с картинкой), с какого года увлекаетесь.
3. \* Напишите скрипт, заполняющий БД тестовыми данными. Сделайте любыми доступными вам средствами. В следующих уроках мы рассмотрим как это сделать, а пока поробуйте сами.

Все проблемы и пути их решения, подробно обсудим на следующем занятии.

# Дополнительные материалы

Все то, о чём сказано в методичке, но подробнее:

1. [Модель User](http://djbook.ru/rel1.8/ref/contrib/auth.html#django.contrib.auth.models.User)
2. [Выполнение запросов](http://djbook.ru/rel1.7/topics/db/queries.html#making-queries)
3. [Типы полей](http://djbook.ru/rel1.7/ref/models/fields.html#field-types)
4. [Параметры полей](http://djbook.ru/rel1.7/ref/models/fields.html#module-django.db.models.fields)
5. [Миграции](http://djbook.ru/rel1.8/topics/migrations.html)

# Используемая литература

Для подготовки данного методического пособия были использованы следующие ресурсы:

1. [Теория баз данных](http://www.site-do.ru/db/db1.php)
2. [Перевод документации](http://djbook.ru/)