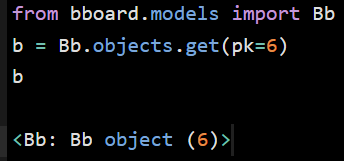
**Тема занятия № 9: Модуль 6. Запись данных**

1. Правка записей

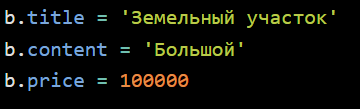
Модели призваны упростить работу с данными, хранящимися в информационной базе. В том числе облегчить запись данных в базу.

Проще всего исправить уже имеющуюся в модели запись. Для этого нужно извлечь ее каким-либо образом:



Здесь мы в консоли Django извлекаем объявление с ключом б.

Занести в поля извлеченной записи новые значения можно, просто присвоив их атрибутам класса модели, представляющим эти поля:



После этого останется выполнить сохранение записи, вызвав метод save () модели:

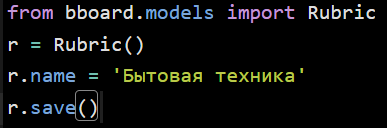


Поскольку эта запись имеет ключ (ее ключевое поле заполнено), Django сразу узнает, что она уже была ранее сохранена в базе, и выполнит обновление, отправив СУБД SQL-команду update.

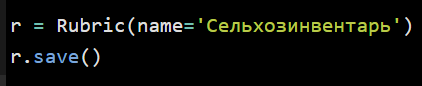
1. Создание записей

Создать новую запись в модели можно тремя способами:

□ создать новый экземпляр класса модели, вызвав конструктор без параметров, занести в поля нужные значения и сохранить запись, вызвав у нее метод save ():

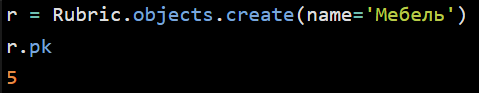


□ создать новый экземпляр класса модели, указав значения полей в вызове конструктора — через одноименные параметры, и сохранить запись:



□ все классы моделей поддерживают атрибут objects, в котором хранится диспетчер записей — объект, представляющий все хранящиеся в модели записи и являющийся экземпляром класса Manager ИЗ модуля django.db.models.

Класс Manager поддерживает метод create (), который принимает с именованными параметрами значения полей создаваемой записи, создает эту запись, сразу же сохраняет и возвращает в качестве результата. Вот пример использования этого метода:



Удостовериться в том, сохранена ли запись, можно, запросив значение ее ключевого поля (оно всегда доступно через универсальный атрибут класса рк). Если оно хранит значение, значит, запись была сохранена.

При создании новой записи любым из описанных ранее способов Django проверяет значение ее ключевого поля. Если таковое хранит пустую строку или None (т.е. Ключ отсутствует), фреймворк вполне резонно предполагает, что запись нужно добавить в базу, и выполняет ее добавление посылкой СУБД SQL-команды insert. Если уж зашла речь о диспетчере записей Manager, то нужно рассказать еще о паре полезных методов, которые он поддерживает:

□ get\_or\_create (<набор фильтров>[, defaults=None] ) — ищет запись на основе заданного набора фильтров. Если подходящая запись не будет найдена, метод создаст и сохранит ее, использовав набор фильтров для указания значений полей новой записи.

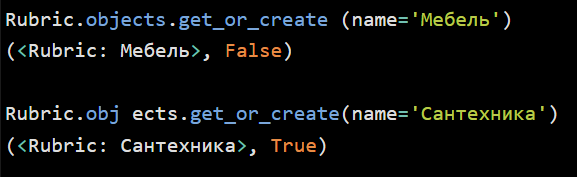
Необязательному параметру defaults можно присвоить словарь, указывающий значения для остальных полей создаваемой записи (подразумевается, что модель не содержит поля с именем defaults. Если же такое поле есть и по нему нужно ВЫПОЛНИТЬ ПОИСК, ТО следует использовать фильтр вида defaults\_\_exact).

В качестве результата метод возвращает кортеж из двух значений:

• записи модели, найденной в базе или созданной только что;

• True, если эта запись была создана, или False, если она была найдена в базе.

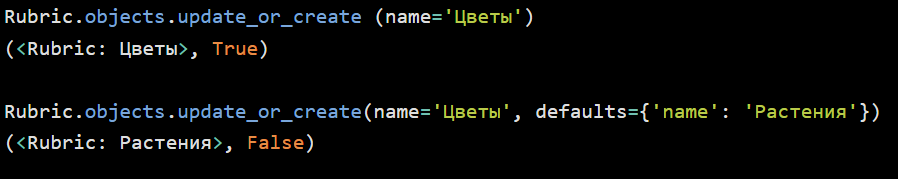
Пример:



Внимание!

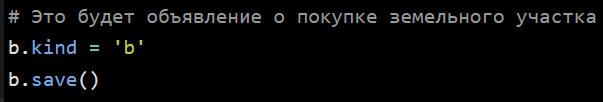
Метод get or create () способен вернуть только одну запись, удовлетворяющую заданным критериям поиска. Если таких записей в модели окажется более одной, то будет возбуждено исключение multipleobjectsreturned ИЗ МОДУЛЯ django. Core, exceptions.

□ update\_or\_create (<набор фильтров>[ f defaults=None]) — аналогичен методу Get or create (), но в случае, если запись найдена, заносит в ее поля новые значения, заданные в словаре, который указан в параметре defaults. Пример:

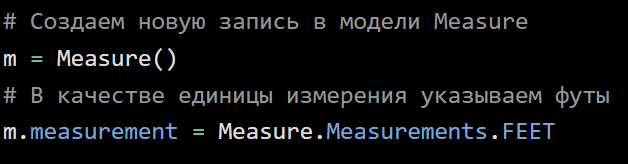


1. Занесение значений в поля со списком

В поле со списком, в котором перечень доступных значений представлен последовательностью Python, следует заносить внутреннее значение, предназначенное для записи в поле:

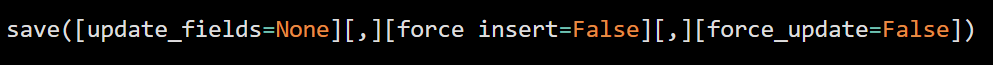


Если же перечень значений представлен перечислением, то в поле нужно заносить элемент перечисления:

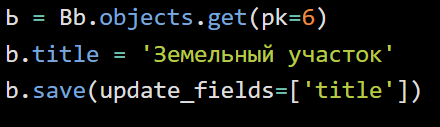


1. Метод save()

Формат вызова метода save () модели, сохраняющего запись:



Необязательный параметр update fields указывает последовательность имен полей модели, которые нужно обновить. Его имеет смысл задавать только при обновлении записи, если были изменены значения не всех, а одного или двух полей, и если поля, не подвергшиеся обновлению, хранят объемистые данные (например, большой текст в поле текстового типа). Пример:



Если параметр update fields не задан, будут обновлены все поля модели.

Если параметрам force insert и force update задать значение False, то Django сам будет принимать решение, создать новую запись или обновить имеющуюся (на основе чего он принимает такое решение, мы уже знаем). Однако мы можем явно указать ему создать или изменить запись, присвоив значение True параметру Force insert ИЛИ force update соответственно.

Такое может пригодиться, например, в случае, если какая-то таблица содержит ключевое поле не целочисленного автоинкрементного, а какого-то иного типа, например, строкового. Значение в такое поле при создании записи придется заносить вручную, но при сохранении записи, выяснив, что ключевое поле содержит значение, Django решит, что эта запись уже была сохранена ранее, и попытается обновить ее, что приведет к ошибке. Чтобы исключить такую ситуацию, при вызове метода save () следует задать параметр force insert со значением True.

Здесь нужно иметь в виду две особенности. Во-первых, задание списка обновляемых полей в параметре update fields автоматически дает параметру force update значение True (т. Е. Явно указывает обновить запись). Во-вторых, указание True для обоих описанных ранее параметров вызовет ошибку.