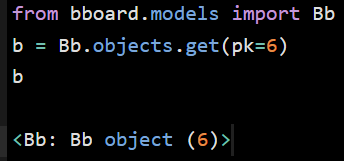
**Тема занятия № 6: Модуль 6. Запись и выборка данных**

1. Правка записей

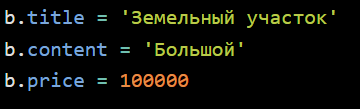
Модели призваны упростить работу с данными, хранящимися в информационной базе. В том числе облегчить запись данных в базу.

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:

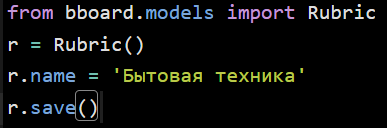


Поскольку эта запись имеет ключ (ее ключевое поле заполнено), Django сразу узнает, что она уже была ранее сохранена в базе, и выполнит обновление, отправив СУБД SQL-команду update.

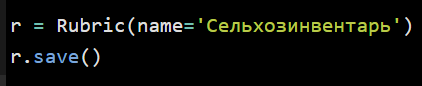
1. Создание записей

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 ():

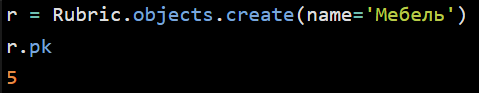


□ создать новый экземпляр класса модели, указав значения полей в вызове конструктора — через одноименные параметры, и сохранить запись:



□ 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.

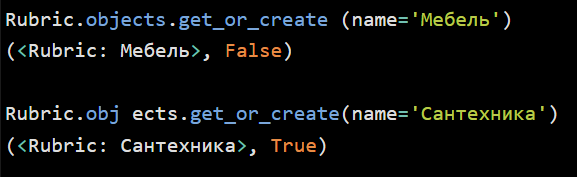
Необязательному параметру defaults можно присвоить словарь, указывающий значения для остальных полей создаваемой записи (подразумевается, что модель не содержит поля с именем defaults. Если же такое поле есть и по нему нужно ВЫПОЛНИТЬ ПОИСК, ТО следует использовать фильтр вида defaults\_\_exact).

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.

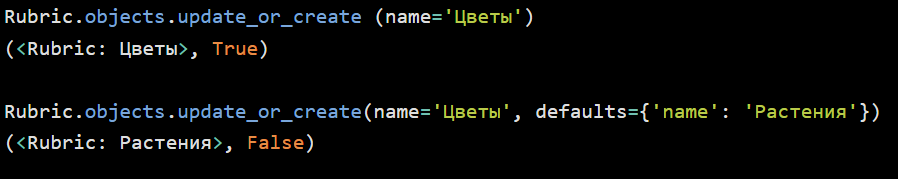
Пример:



**Внимание!**

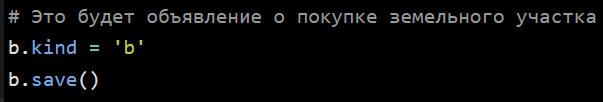
Метод get or create () способен вернуть только одну запись, удовлетворяющую заданным критериям поиска. Если таких записей в модели окажется более одной, то будет возбуждено исключение multipleobjectsreturned ИЗ МОДУЛЯ django. Core, exceptions.

□ 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:

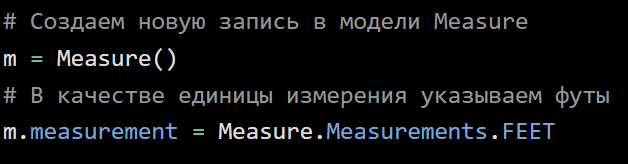


1. Занесение значений в поля со списком

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

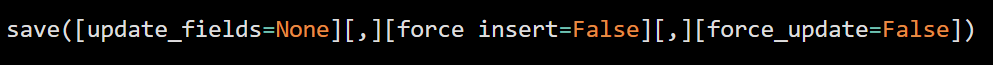


Если же перечень значений представлен перечислением, то в поле нужно заносить элемент перечисления:

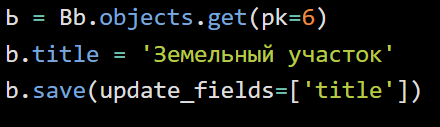


1. Метод save()

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.

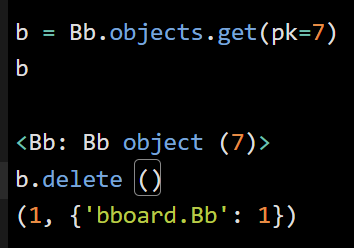
Если параметрам force insert и force update задать значение False, то Django сам будет принимать решение, создать новую запись или обновить имеющуюся (на основе чего он принимает такое решение, мы уже знаем). Однако мы можем явно указать ему создать или изменить запись, присвоив значение True параметру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

1. Удаление записей

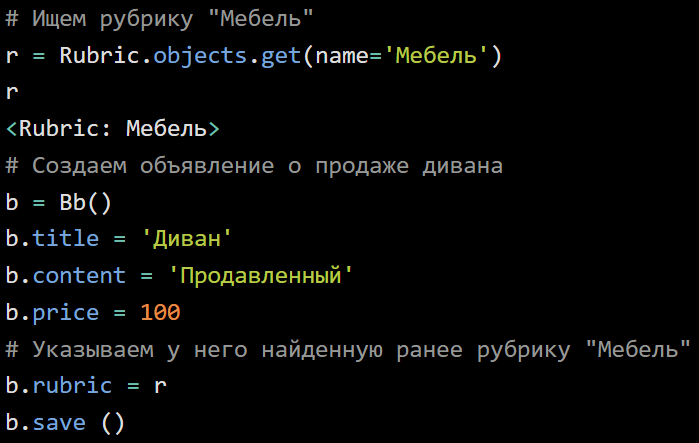
Для удаления записи достаточно вызвать у нее метод delete ():



Метод delete о возвращает в качестве результата кортеж. Его первым элементом станет количество удаленных записей во всех моделях, имеющихся в проекте. Вторым элементом является словарь, в котором ключи элементов представляют отдельные модели, а их значения — количество удаленных из них записей. Особой практической ценности этот результат не представляет.

1. Обработка связанных записей

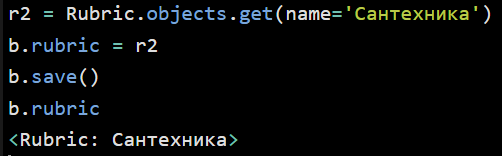
Django предоставляет ряд инструментов для удобной работы со связанными записями: создания, установления и удаления связи.Связать запись вторичной модели с записью первичной модели можно, присвоив полю внешнего ключа в записи вторичной модели нужный объект-запись первичной модели, например:



Указать объект-запись первичной модели можно и в вызове метода create о диспетчера записей:



Таким же образом выполняется связывание записи вторичной модели с другой записью первичной таблицы:



Модель, представляющая запись первичной таблицы, получает атрибут с именем вида <имя связанной вторичной модели>\_set. Он хранит экземпляр класса Relatedmanager ИЗ модуля django.db.models, fields, related, представляющий набор связанных записей вторичной таблицы и называемый диспетчером обратной связи.

**Внимание!**

Описанный ранее атрибут класса получает имя <имя связанной вторичной модели>\_Set по умолчанию. Однако это имя можно изменить при объявлении поля внешнего ключа, указав его в параметре reiated name конструктора класса поля.

Класс reiatedmanager поддерживает два очень полезных метода:

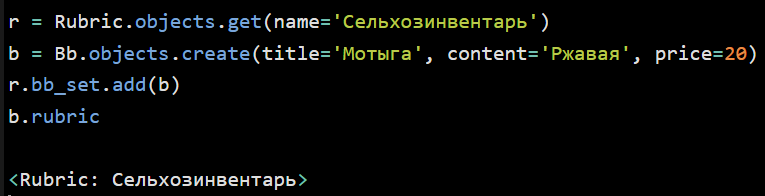
О add(<связываемая запись 1>, <связываемая запись 2> . . . < связываемая запись n>[, buik=True]) — связывает с текущей записью первичной модели записи вторичной модели, переданные в качестве параметров.

Если значение параметра bulk равно True, то записи будут связаны непосредственно отдачей СУБД SQL-команды, без манипуляций с объектами моделей, представляющих связываемые записи. Это поведение по умолчанию, и оно позволяет увеличить производительность.

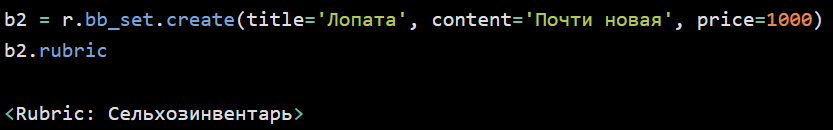
Если значение параметра bulk равно False, то записи будут связываться посредством манипуляций объектами модели, представляющих связываемые записи. Это может пригодиться, если класс модели содержит переопределенные методы Save() И delete().

К моменту вызова метода add () текущая запись первичной модели должна быть сохранена. Не забываем, что в поле внешнего ключа записи вторичной модели сохраняется ключ записи первичной модели, а он может быть получен только после сохранения записи (если в модели используется стандартное ключевое поле целочисленного автоинкрементного типа).

Пример:

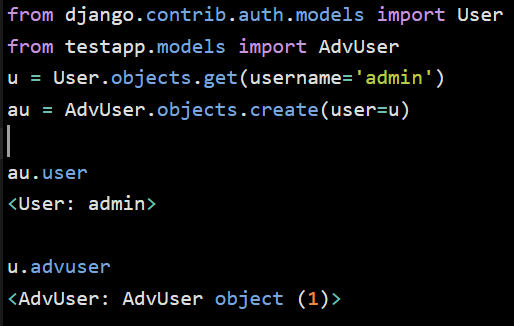


□ create () — метод унаследован от класса Manager и, помимо создания записи вторичной модели, также выполняет ее связывание с текущей записью первичной модели:

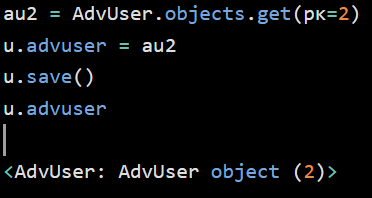


Связь такого рода очень проста, соответственно, программных инструментов для ее установления Django предоставляет немного.

Связать записи вторичной и первичной модели можно, присвоив запись первичной модели полю внешнего ключа записи вторичной модели. Вот пример создания записи вторичной модели Advuser и связывания ее с записью первичной модели user, представляющей пользователя admin:



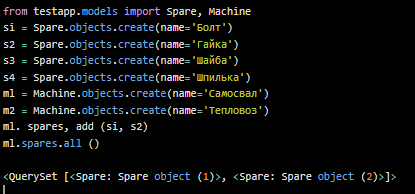
Первичная модель при этом получит атрибут, хранящий связанную запись вторичной модели. Имя этого атрибута совпадет с именем вторичной модели. Следовательно, связать записи первичной и вторичной модели можно, присвоив запись вторичной модели описанному ранее атрибуту. Вот пример связывания записи модели User С другой записью модели advuser:

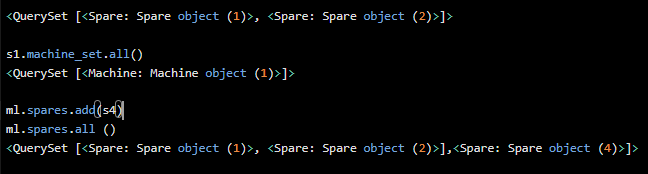


Если между двумя моделями была установлена связь такого рода, то перед собственно связыванием записей нам обязательно нужно их сохранить.

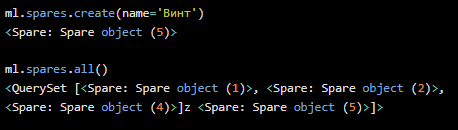
В случае связей "один-со-многими" и "один-с-одним" поле внешнего ключа, объявленное во вторичной модели, всегда хранит непосредственно объект первичной модели, представляющий связанную запись. Но в случае связи ”многие-со-многими" это не так— атрибут, представляющий поле, хранит экземпляр класса Relatedmanager —диспетчер обратной СВЯЗИ.

□ add () — для добавления указанных записей в число связанных с текущей записью:





□ create о — для создания новых записей связанной модели и одновременного связывания их с текущей записью:



□ set (^последовательность связываемых записей>[, bulk=True] [, clear=False]) — то же самое, что add о, но не добавляет указанные записи в число связанных с текущей записью, а заменяет ими те, что были связаны с ней ранее.

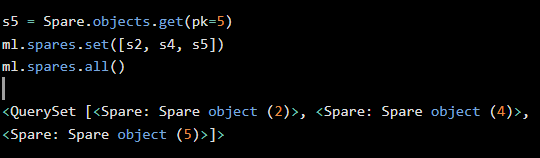
Если значение параметра bulk равно True, то записи будут связаны непосредственно отдачей СУБД SQL-команды, без манипуляций с объектами моделей, представляющих связываемые записи. Это поведение по умолчанию, и оно позволяет увеличить производительность.

Если значение параметра bulk равно False, то записи будут связываться посредством манипуляций объектами модели, представляющих связываемые записи. Это может пригодиться, если класс модели содержит переопределенные методы Save() И delete().

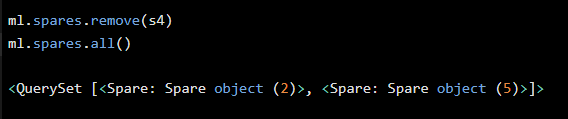
Если значение параметра clear равно True, то Django сначала очистит список связанных записей, а потом свяжет заданные в методе записи с текущей записью.

Если же его значение равно False, то указанные записи, отсутствующие в списке связанных, будут добавлены в него, а связанные записи, отсутствующие в последовательности указанных в вызове метода, — удалены из списка связанных (поведение по умолчанию).

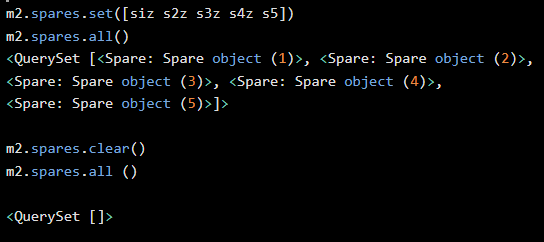
Пример:



О remove (<удаляемая запись 1>, <удаляемая запись 2> . . . <удаляемая запись п>) — удаляет указанные записи из списка связанных с текущей записью:



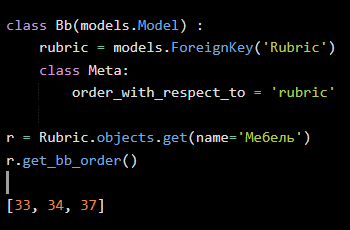
□ clear () —полностью очищает список записей, связанных с текущей:

****

1. Произвольное переупорядочивание записей

Если у вторичной модели был указан параметр order with respect to, то ее записи, связанные с какой-либо записью первичной модели, могут быть произвольно переупорядочены (подробности — в разд, 4.4). Для этого следует получить запись первичной модели и вызвать у нее нужный метод:

□ get\_<HMH вторичной модели>\_отдет () — возвращает список ключей записей вторичной модели, связанных с текущей записью первичной модели:



□ set\_<имя вторичной модели>\_от<^ет (<список ключей записей вторичной модели>) — задает новый порядок следования записей вторичной модели. В качестве параметра указывается список ключей записей, в котором ключи должны быть выстроены в нужном порядке. Пример:



1. Массовые добавление, правка и удаление записей

Если возникает необходимость создать, исправить или удалить сразу большое количество записей, то удобнее использовать средства Django для массовой записи данных. Это следующие методы класса Manager (они также поддерживаются производным ОТ него классом relatedmanager):

□ bulk\_create ^последовательность добавляемых записей>[, batch\_size=None] [, ignore\_Conflicts=False]) — добавляет В модель записи, указанные В последовательности.

Параметр batch size задает количество записей, которые будут добавлены в одной SQL-команде. Если он не указан, все заданные записи будут добавлены в одной команде (поведение по умолчанию).

Параметр ignore\_conflicts появился в Django 2.2. Если ему задать значение False, то при попытке добавить записи, нарушающие заданные в таблице условия (например, содержащие неуникальные значения), будет возбуждено исключение integrityerror (поведение по умолчанию). Если же параметр получит значение True, исключения генерироваться не будут, а записи, нарушающие условия, просто не добавятся в таблицу.

Внимание!

Некоторые СУБД, в частности Oracle, не поддерживают игнорирование нарушений условий, записанных в таблице, при добавлении записей. В таком случае при попытке вызвать метод bulk create с указанием у параметра ignore conflicts значения True будет возбуждено исключение notsupportederror ИЗ модуля django.db.utils.

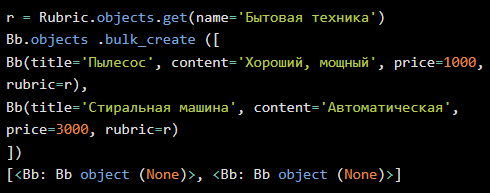
В качестве результата возвращается набор добавленных в модель записей, представленный экземпляром класса queryset.

**Внимание!**

Метод bulk create () не создает объекты модели, а непосредственно отправляет СУБД SQL-команду, создающую записи. Поэтому в записях, возвращенных им, ключевое поле не заполнено.

Исходя из этого, нужно помнить, что метод save () у добавляемых таким образом записей не вызывается. Если он переопределен в модели, чтобы осуществить при сохранении какие-либо дополнительные действия, то эти действия выполнены не будут.

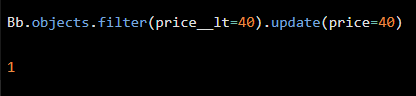
Пример:



□ update {<новые значения полей>) — исправляет все записи в наборе, задавая для них новые значения полей. Эти значения задаются в параметрах метода, одноименных с нужными полями модели. В качестве результата возвращается количество исправленных записей.

Метод save () у исправляемых записей не вызывается, что может быть критично, если последний переопределен и выполняет какие-либо дополнительные действия.

Запишем в объявления, в которых указана цена меньше 40 руб., цену 40 руб.:



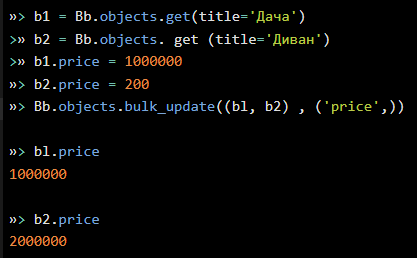
□ bulk\_update (<записи>, <поля>[, batch\_size=None]) (начиная C Django 2.2)—позволяет исправить произвольное число записей.

Исправляемые записи выбираются из модели, после чего в их поля заносятся новые значения путем присваивания соответствующим атрибутам класса модели. Последовательность исправленных таким образом записей передается методу первым параметром. Вторым параметром указывается последовательность из имен полей, значения которых были исправлены.

Параметр batch size задает число записей, которые будут исправлены в одной SQL-команде. Если он не указан, все заданные записи будут исправлены в одной команде (поведение по умолчанию).

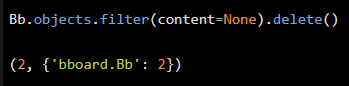
Метод save () у исправляемых записей в этом случае также не вызывается.

Исправим цены дачи и дивана на 1 000 000 и 200 соответственно:



□ delete () — удаляет все записи в наборе. В качестве результата возвращает словарь, аналогичный таковому, возвращаемому методом delete о модели. Метод delete о у удаляемых записей не вызывается, что может быть критично, если последний переопределен.

Удалим все объявления, в которых не было указано описание товара:



Методы для массовой записи данных работают быстрее, чем программные инструменты моделей, поскольку напрямую ’’общаются” с базой данных. Однако не забываем, что при их использовании дополнительные операции, выполняемые моделями (в частности, автоматическое получение ключей записей и выполнение методов Save () И delete () ), не работают.

1. Выполнение валидации модели

Валидация непосредственно модели выполняется редко — обычно это делается на уровне формы, связанной с ней. Но на всякий случай выясним, как провести ее.

Валидацию модели запускает метод full clean ():



Параметр exclude задает последовательность имен полей, значения которых проверяться не будут. Если он опущен, будут проверяться все поля.

Если параметру vaiidate\_unique присвоить значение True, то при наличии в модели уникальных полей также будет проверяться уникальность заносимых в них значений (поведение по умолчанию). Если значение этого параметра— False, такая проверка проводиться не будет.

Метод не возвращает никакого результата. Если в модель занесены некорректные данные, ТО она возбуждает исключение validationerror ИЗ модуля django.core, Exceptions.

В последнем случае в атрибуте message\_dict модели будет храниться словарь с сообщениями об ошибках. Ключи элементов будут соответствовать полям модели, а значениями элементов станут списки с сообщениями об ошибках.

Примеры:

