**Тема занятия № 4: Модели. Базовые инструменты. Миграции**

1. Объявление моделей

Модели объявляются на уровне отдельного приложения, в модуле models.py пакета Этого приложения.

Класс модели должен быть производным от класса Model из модуля Django.db.models. Также имеется возможность сделать класс модели производным от другого класса модели. Чтобы модели были успешно обработаны программным ядром Django, содержащее Их приложение должно быть зарегистрировано в списке приложений проекта.

Модель может быть создана для представления как еще не существующей в базе Таблицы (тогда для ее создания следует сгенерировать миграцию), так и уже Имеющейся (в этом случае при объявлении модели придется дополнительно указать имя таблицы и имена всех входящих в нее полей).

1. Объявление полей модели

Для представления отдельного поля таблицы в модели создается атрибут класса, Которому присваивается экземпляр класса, представляющего поле нужного типа (строкового, целочисленного, логического и т. Д.). Дополнительные параметры создаваемого поля указываются в соответствующих им именованных параметрах конструктора класса поля.

Параметры, поддерживаемые полями всех типов:

Verbose name — "человеческое” название поля, которое будет выводиться на вебстраницах. Если не указано, то будет выводиться имя поля;

□ help text— дополнительный поясняющий текст, выводимый на экран. По Умолчанию — пустая строка.

Содержащиеся в этом тексте специальные символы HTML не преобразуются в литералы и выводятся как есть. Это позволяет отформатировать поясняющий текст HTML-тегами;

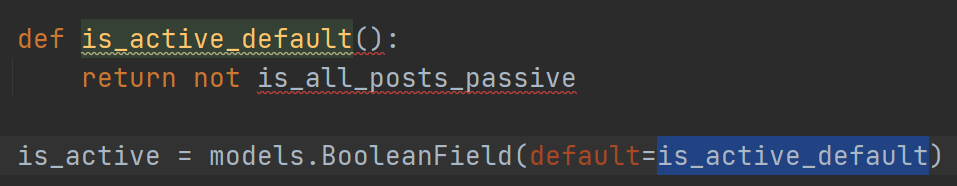
□ default — значение по умолчанию, записываемое в поле, если в него явно не было занесено никакого значения. Может быть указано двумя способами:

• как обычное значение любого неизменяемого типа:



Если в качестве значения по умолчанию должно выступать значение изменяемого типа (список или словарь Python), то для его указания следует использовать второй способ;

• как ссылка на функцию, вызываемую при создании каждой новой записи и возвращающую в качестве результата заносимое в поле значение:

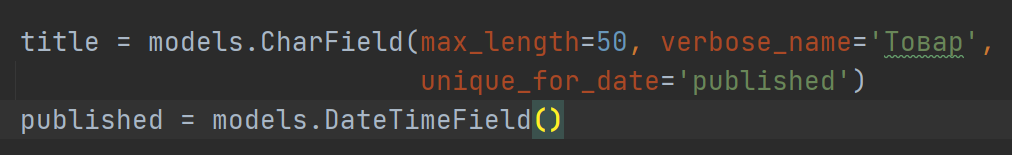


□ unique — если True, то в текущее поле может быть занесено только уникальное в пределах таблицы значение (уникальное поле). При попытке занести значение, уже имеющееся в том же поле другой записи, будет возбуждено исключение Integrityerror из модуля django.db.

Если поле помечено как уникальное, по нему автоматически будет создан индекс. Поэтому явно задавать для него индекс не нужно.

Если False, то текущее поле может хранить любые значения. Значение по умолчанию — False;

□ unique for date — если в этом параметре указать представленное в виде строки имя поля даты (datefieid) или даты и времени (datetimefieid), то текущее поле может хранить только значения, уникальные в пределах даты, которая хранится в указанном поле. Пример:



В этом случае Django позволит сохранить в поле title только значения, уникальные в пределах даты, хранящейся в поле published;

□ unique for month — то же самое, что и unique for date, но в расчет принимается не всё значение даты, а лишь месяц;

□ unique fог уеаг — то же самое, что и unique for date, но в расчет принимается не всё значение даты, а лишь год;

□ null — если True, то поле в таблице базы данных может хранить значение null и, таким образом, являться необязательным к заполнению. Если False, то поле в таблице должно иметь какое-либо значение, хотя бы пустую строку. Значение ПО умолчанию — False.

Отметим, что у строковых и текстовых полей, даже обязательных к заполнению (т.е. При их добъявлении параметру null было присвоено значение False), вполне допустимое значение— пустая строка. Если сделать поля необязательными к заполнению, задав True для параметра null, то они вдобавок к этому могут хранить значение null. Оба значения фактически представляют отсутствие каких-либо данных в поле, и эту ситуацию придется как-то обрабатывать. Поэтому, чтобы упростить обработку отсутствия значения в таком поле, его не стоит делать необязательным к заполнению.

Параметр null затрагивает только поле таблицы, но не поведение Django. Даже если какое-то поле присвоением параметру значения True было помечено как необязательное, фреймворк по умолчанию все равно не позволит занести в него пустое значение;

□ blank — если True, то Django позволит занести в поле пустое значение, тем самым сделав поле необязательным к заполнению, если False — не позволит. По умолчанию — False.

Параметр blank задает поведение самого фреймворка при выводе на экран вебформ и проверке введенных в них данных. Если этому параметру дано значение True, то Django позволит занести в поле пустое значение (например, для строкового поля — пустую строку), даже если это поле было помечено как обязательное к заполнению (параметру null было дано значение False);

□ db index — если True, то по текущему полю в таблице будет создан индекс, если False — не будет. По умолчанию — False;

□ primary key — если True, то текущее поле станет ключевым. Такое поле будет помечено как обязательное к заполнению и уникальное (параметру null неявно будет присвоено значение False, а параметру unique — True), и по нему будет создан ключевой индекс.

Внимание! В модели может присутствовать только одно ключевое поле.

Если False, то поле не будет преобразовано в ключевое. Значение по умолчанию — False.

Если ключевое поле в модели не было задано явно, сам фреймворк создаст в ней целочисленное автоинкрементное ключевое поле с именем id;

□ editable — если True, то поле будет выводиться на экран в составе формы, если False —не будет (даже если явно создать его в форме). По умолчанию — True;

□ db coiumn — имя поля таблицы в виде строки. Если не указано, то поле таблицы получит то же имя, что и поле модели.

Например, в таблице, представляемой моделью вь приложения bboard будут созданы поля id (ключевое поле неявно формируется самим фреймворком), title, content, price И published.

Здесь приведены не все параметры, поддерживаемые конструкторами классов полей. Некоторые параметры мы изучим позже.

**Классы полей моделей**

Все классы полей, поддерживаемые Django, объявлены в модуле django.db.models. Каждое такое поле позволяет хранить значения определенного типа. Многие типы полей поддерживают дополнительные параметры, также описанные далее.

□ charfield— строковое поле, хранящее строку ограниченной длины. Занимает в базе данных объем, необходимый для хранения числа символов, указанного в качестве размера этого поля. Поэтому, по возможности, лучше не создавать в моделях строковые поля большой длины.

Обязательный параметр max iength указывает максимальную длину заносимого в поле значения в виде целого числа в символах.

□ textfield— текстовое поле, хранящее строку неограниченной длины. Такие поля рекомендуется применять для сохранения больших текстов, длина которых заранее не известна и может варьироваться.

Необязательный параметр max length задает максимальную длину заносимого в поле текста. Если не указан, то в поле можно записать значение любой длины.

□ emailfieid —адрес электронной почты в строковом виде.

Необязательный параметр max length указывает максимальную длину заносимого в поле адреса в виде целого числа в символах. Значение по умолчанию: 254.

□ urlfieid —интернет-адрес.

Необязательный параметр max length указывает максимальную длину заносимого в поле интернет-адреса в виде целого числа в символах. Значение по умолчанию: 200.

□ siugfieid — слаг, т. Е. Строка, однозначно идентифицирующая запись и включаемая в состав интернет-адреса. Поддерживает два необязательных параметра:

• max length — максимальная длина заносимой в поле строки в символах. По умолчанию: 50;

• aiiow unicode — если True, то хранящийся в поле слаг может содержать любые символы Unicode, если False — только символы из кодировки ASCII. По умолчанию — False.

Для каждого поля такого типа автоматически создается индекс, поэтому указывать параметр db index со значением True нет нужды.

□ booleanfield —логическое поле, хранящее значение True ИЛИ False.

Внимание! Значение поля booleanfield ПО умолчанию— None, а не False, как можно было бы предположить.

Для поля этого типа можно указать параметр null со значением True, в результате чего оно получит возможность хранить еще и значение null.

□ nuiibooieanfieid—то же самое, что booleanfield, но дополнительно позволяет хранить значение null. Этот тип поля оставлен для совместимости с более старыми версиями Django, и использовать его во вновь разрабатываемых проектах не рекомендуется.

□ integerfieid —знаковое целочисленное поле обычной длины (32-разрядное).

□ smaiiintegerfieid— знаковое целочисленное поле половинной длины (16-разрядное).

□ bigintegerfieid— знаковое целочисленное значение двойной длины (64-разрядное).

□ positiveintegerfieid —беззнаковое целочисленное поле обычной длины (32-разрядное).

□ positivesmaiiintegerfieid— беззнаковое целочисленное поле половинной длины (16-разрядное).

□ fioatfieid —вещественное число.

□ decimaifieid— вещественное число фиксированной точности, представленное объектом типа Decimal из модуля decimal Python. Поддерживает два обязательных параметра:

• max digits —максимальное количество цифр в числе;

• decimal places —количество цифр в дробной части числа.

Пример объявления поля для хранения чисел с шестью цифрами в целой части и двумя — в дробной:



□ datefieid —значение даты в виде объекта типа date из модуля datetime Python.

Класс datefieid поддерживает два необязательных параметра:

• auto now — если True, то при каждом сохранении записи в поле будет заноситься текущее значение даты. Это может использоваться для хранения даты последнего изменения записи.

Если False, то хранящееся в поле значение при сохранении записи никак не затрагивается. Значение по умолчанию — False;

• auto now add — то же самое, что auto now, но текущая дата заносится в поле только при создании записи и при последующих сохранениях не изменяется. Может пригодиться для хранения даты создания записи. Указание значения True для любого из этих параметров приводит к тому, что поле становится невидимым и необязательным для заполнения на уровне Django (т.е. Параметру editable присваивается False, а параметру blank —True).

□ datetimefieid — то же самое, что и datefieid, но хранит значение временной отметки в виде объекта типа datetime ИЗ модуля datetime.

□ timefieid— значение времени в виде объекта типа time из модуля datetime Python. Поддерживаются необязательные параметры auto now и auto now add (см. Описание класса datefieid).

□ durationfieid —промежуток времени, представленный объектом типа timedelta из модуля datetime Python.

□ binaryfieid— двоичные данные произвольной длины. Значение этого поля представляется объектом типа bytes.

□ genericipaddressfieid— IP-адрес, записанный для протокола ipv4 или ipv6, в виде строки. Поддерживает два необязательных параметра:

• protocol — обозначение допустимого протокола для записи IP-адресов, представленное в виде строки. Поддерживаются значения "ipv4”, "ipv6" и "both" (поддерживаются оба протокола). По умолчанию — "both";

• inpack\_ipv4 — если True, то IP-адреса протокола ipv4, записанные в формате Ipv6, будут преобразованы к виду, применяемому в ipv4. Если False, то такое преобразование не выполняется. Значение по умолчанию — False. Этот параметр принимается во внимание только в том случае, если для параметра Protocol указано значение "both".

□ autofield — автоинкрементное поле. Хранит уникальные, постоянно увеличивающиеся целочисленные значения обычной длины (32-разрядные). Практически всегда используется в качестве ключевого поля.

Как правило, нет необходимости объявлять такое поле явно. Если модель не содержит явно объявленного ключевого поля любого типа, то Django самостоятельно создаст ключевое поле типа autofield.

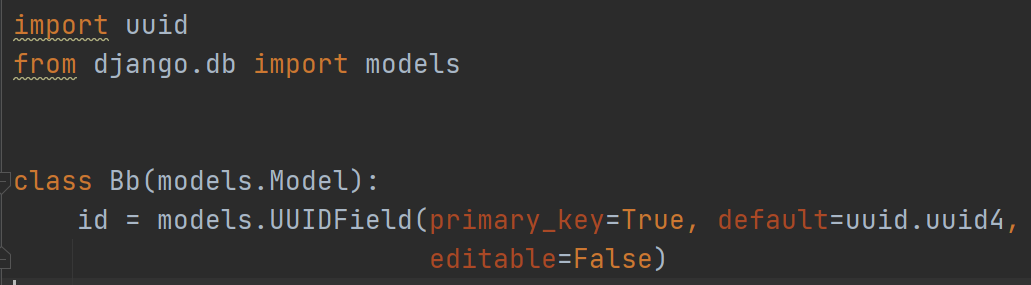
□ smallautofield (начиная С django 3.0)— ТО же самое, ЧТО autofield, но хранит целочисленное значение половинной длины (16-разрядное).

□ bigautofieid — то же самое, что autofield, но хранит целочисленное значение двойной длины (64-разрядное).

П uuidfieid — уникальный универсальный идентификатор, представленный объектом типа uuid из модуля uuid Python, в виде строки.

Поле такого типа может использоваться в качестве ключевого вместо поля Autofield, smallautofieid или bigautofieid. Единственный недостаток: придется генерировать идентификаторы для создаваемых записей самостоятельно.

ВОТ Пример Объявления ПОЛЯ uuidfield:



**Создание полей со списком**

Поле со списком способно хранить значение из ограниченного набора, заданного в особом перечне. Поле со списком может иметь любой тип, но наиболее часто применяются строковые и целочисленные поля.

В веб-форме поля со списком представляются в виде раскрывающегося списка, перечисляющего все возможные значения для ввода.

Перечень значений для выбора заносится в параметр choices конструктора поля и может быть задан в виде:

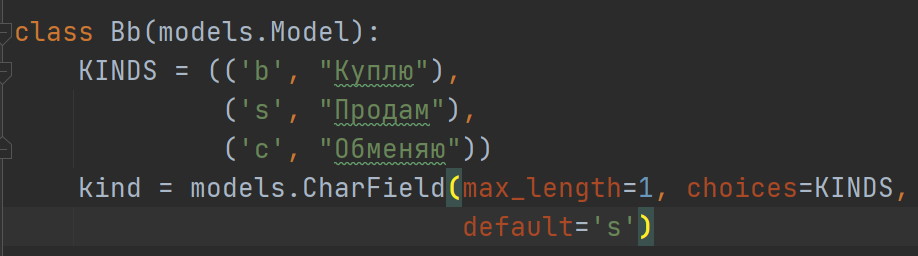
□ последовательности — списка или кортежа, каждый элемент которого представляет отдельное значение и записывается в виде последовательности из двух элементов:

• значения, которое будет непосредственно записано в поле (назовем его внутренним). Оно должно принадлежать к типу, поддерживаемому полем (так, если поле имеет строковый тип, то значение должно представлять собой строку);

• значения, которое будет выводиться в виде пункта раскрывающегося списка (внешнее), должно представлять собой строку.

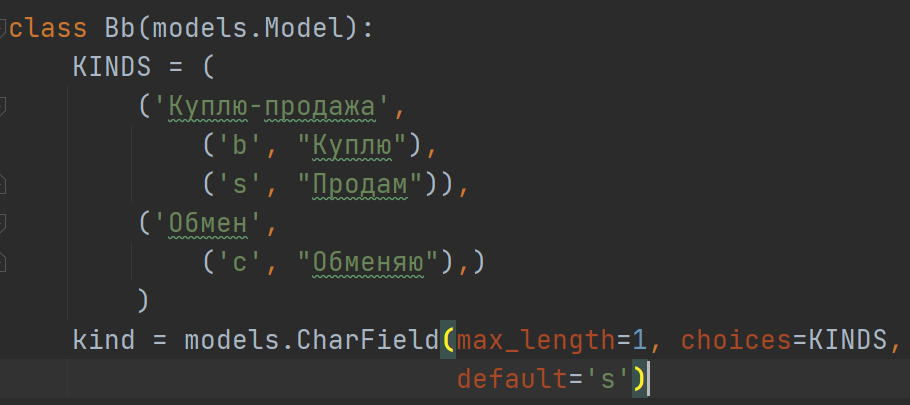
Параметру choices присваивается непосредственно последовательность с перечнем значений.

Пример задания для поля kind списка из трех возможных значений:

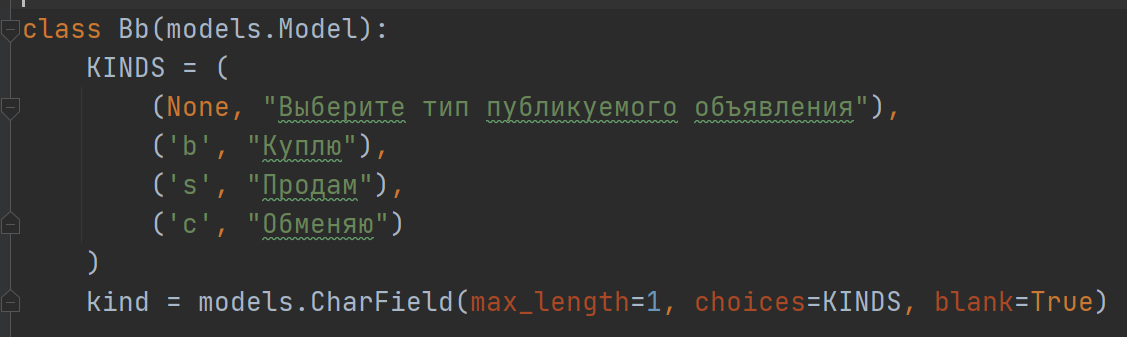
****

Значения из перечня можно объединять в группы. Каждая группа создается последовательностью из двух элементов: текстового названия группы, выводимого на экран, и последовательности из значений описанного ранее формата.

Для примера разобьем перечень позиций kinds на группы ’’Купля-продажа” и ’’Обмен”:



Если поле помечено как необязательное к заполнению на уровне фреймворка (параметру blank конструктора присвоено значение True) или у него не задано значение по умолчанию (отсутствует параметр default), то в списке, перечисляющем доступные для ввода в поле значения, появится пункт----------(9 знаков ’’Минус”), обозначающий отсутствие в поле какого-либо значения. Можно задать для этого пункта другое внешнее значение, добавив в последовательность значений элемент вида (None, ’ <Новое внешнее значение "пустого" пункта>') (если поле имеет строковый тип, вместо None можно поставить пустую строку). Пример:



□ перечисления co строковыми внутренними значениями (теми, что будут непосредственно сохраняться в поле таблицы)— если поле имеет строковый тип (поддерживается, начиная с Django 3.0).

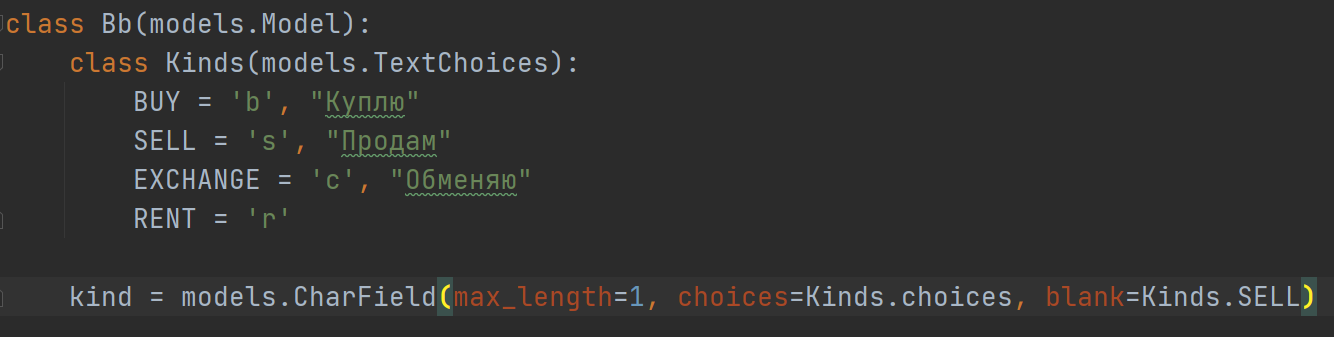
Перечисление должно являться подклассом класса Textchoices из модуля Django.db.models. Каждый элемент перечисления представляет одно из значений, доступных для выбора. В качестве значения элемента можно указать:

• строку— послужит внутренним значением. В качестве внешнего значения будет использовано имя элемента перечисления;

• кортеж из двух строк— первая станет внутренним значением, вторая —внешним.

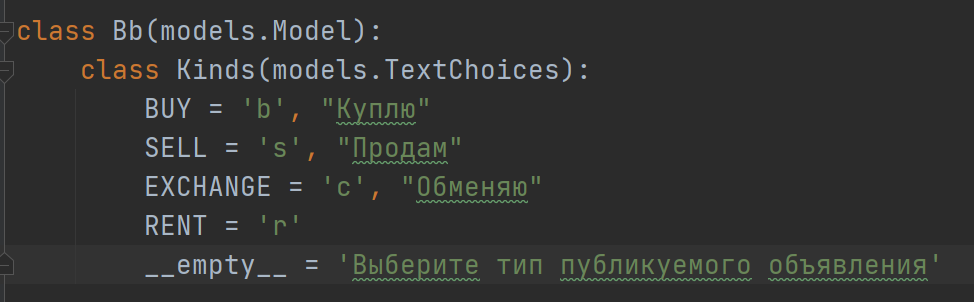
Параметру choices конструктора поля присваивается значение атрибута choices класса перечисления.

Зададим для поля kind перечень значений в виде перечисления:



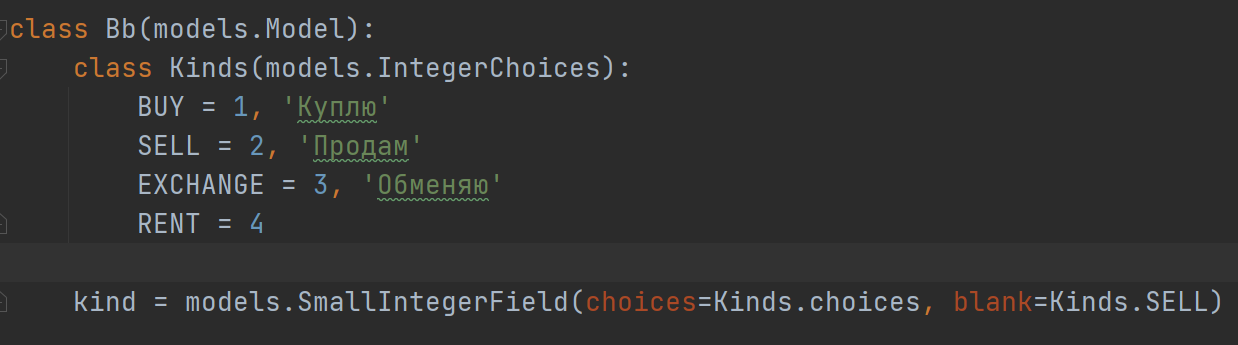
Последний элемент перечисления будет выводиться в списке как Rent, поскольку у него задано лишь внутреннее значение.

Для представления ’’пустого” пункта в перечислении следует создать элемент \_\_empty\_\_и присвоить ему строку с внешним значением:



□ перечисления с целочисленными внутренними значениями — если поле имеет один из целочисленных типов (поддерживается, начиная с Django 3.0).

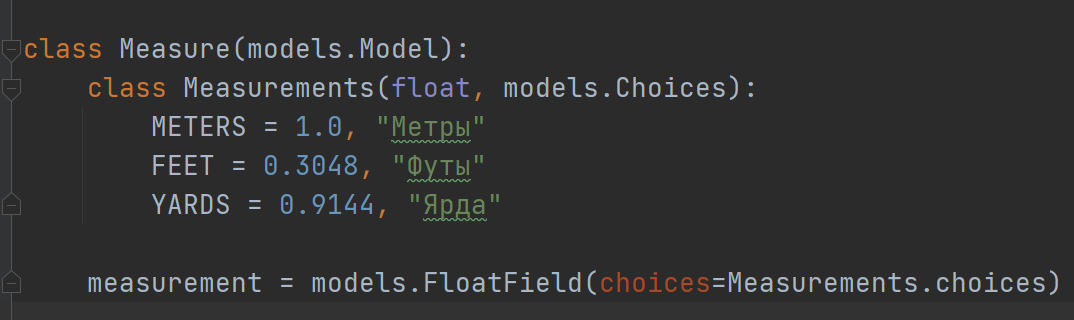
Перечисление должно являться подклассом класса integerchoices из модуля Django.db.modeis (начиная с Django 3.0). В качестве внутренних значений указываются целые числа. В остальном оно аналогично ’’строковому” перечислению, рассмотренному ранее. Пример:



□ перечисления с внутренними значениями произвольного типа — если поле имеет тип, отличный от строкового или целочисленного (появилось в Django 3.0).

Перечисление должно быть подклассом класса, представляющего тип внутренних значений, и класса choices из модуля django.db.models. В начале кортежа, описывающего каждое значение перечня, указываются параметры, передаваемые конструктору класса, что представляет тип внутренних значений. В остальном оно аналогично "строковому” и ’’целочисленному” перечислениям, рассмотренным ранее.

Пример перечисления, содержащего внутренние значения в виде вещественных чисел (тип float):



1. Создание связей между моделями

Связи между моделями создаются объявлением в них полей, формируемых особыми классами ИЗ ТОГО же модуля django.db.models.

Связь ”один-со-многими” связывает одну запись первичной модели с произвольным числом записей вторичной модели. Это наиболее часто применяемый на практике вид связей. Для создания связи такого типа в классе вторичной модели следует объявить поле типа foreignkey. Вот формат конструктора этого класса:

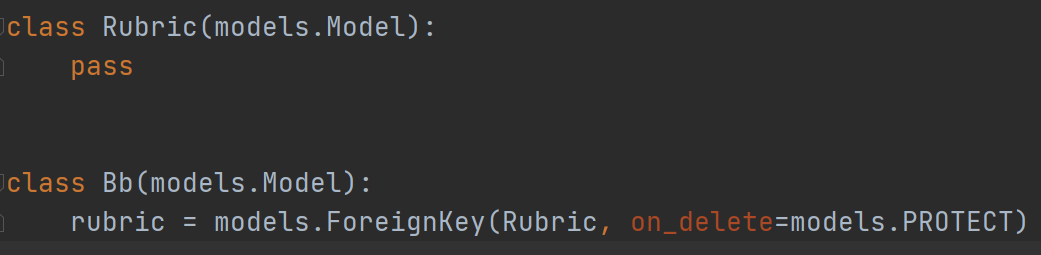
Foreingkey(<связываемая первичная модель>,

Оп\_&е\е^е=<поведение при удалении записи>[,

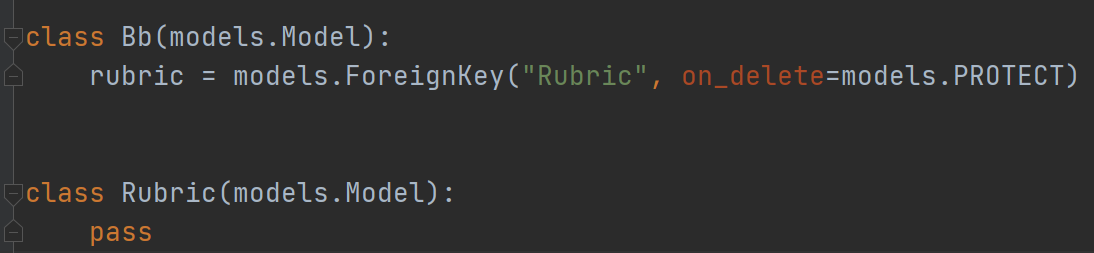
<остальные параметры>] )

Первым, позиционным, параметром указывается связываемая первичная модель в виде:

□ непосредственно ссылки на класс модели, если объявление первичной модели находится перед объявлением вторичной модели (в которой и создается поле внешнего ключа):



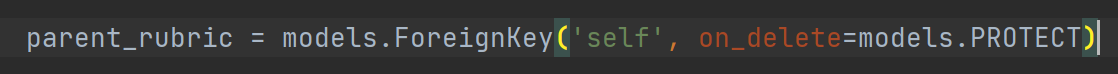
□ строки с именем класса, если первичная модель объявлена после вторичной:



Ссылка на модель из другого приложения проекта записывается в виде строки формата <имя приложения>.<имя класса модели>:



Если нужно создать модель, ссылающуюся на себя (создать рекурсивную связь), то первым параметром конструктору следует передать строку self:



Вторым параметром on delete указывается поведение фреймворка в случае, если будет выполнена попытка удалить запись первичной модели, на которую ссылаются какие-либо записи вторичной модели. Параметру присваивается значение одной из переменных, объявленных в модуле django.db.models:

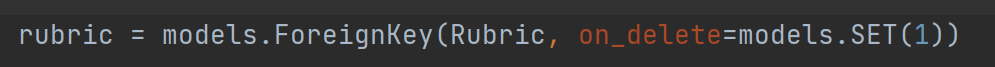
□ cascade — удаляет все связанные записи вторичной модели (каскадное удаление);

□ PROTECT— возбуждает исключение protectederror ИЗ модуля django.db.modeis, тем самым предотвращая удаление записи первичной модели;

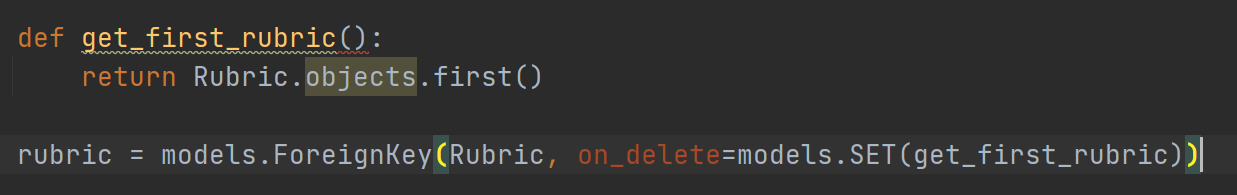
□ set null — заносит в поле внешнего ключа всех связанных записей вторичной модели значение null. Сработает только в том случае, если поле внешнего ключа объявлено необязательным к заполнению на уровне базы данных (параметр Null конструктора поля имеет значение True);

□ set default — заносит в поле внешнего ключа всех связанных записей вторичной модели заданное для него значение по умолчанию. Сработает только в том случае, если у поля внешнего ключа было указано значение по умолчанию (оно задается параметром default конструктора поля);

□ SET (<значение>) —Заносит В поле внешнего ключа указанное значение:



Также можно указать ссылку на функцию, не принимающую параметров и возвращающую значение, которое будет записано в поле:



□ do\_nothing — ничего не делает.

Внимание!

Если СУБД поддерживает межтабличные связи с сохранением ссылочной целостности, то попытка удаления записи первичной модели, с которой связаны записи вторичной модели, в этом случае все равно не увенчается успехом, и будет возбуждено исключение integrityerror из модуля django.db.models.

Полю внешнего ключа рекомендуется давать имя, обозначающее связываемую сущность и записанное в единственном числе. Например, для представления рубрики в модели МЫ объявили поле rubric.

На уровне базы данных поле внешнего ключа модели представляется полем таблицы, имеющим имя вида <имя поля внешнего ключа>\_±д, в веб-форме такое поле будет представляться раскрывающимся списком, содержащим строковые представления записей первичной модели.

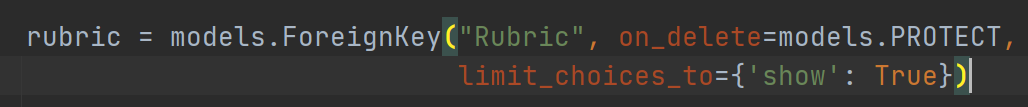
Класс foreignkey поддерживает следующие дополнительные необязательные параметры:

□ limit choices to— позволяет вывести в раскрывающемся списке записей первичной модели, отображаемом в веб-форме, только записи, удовлетворяющие заданным критериям фильтрации.

Критерии фильтрации записываются в виде словаря Python, имена элементов которого совпадают с именами полей первичной модели, по которым должна выполняться фильтрация, а значения элементов укажут значения для этих полей.

Выведены будут записи, удовлетворяющие всем критериям, заданным в таком словаре (т.е. Критерии объединяются по правилу логического И).

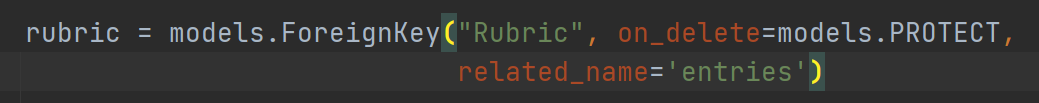
Для примера укажем Django выводить только рубрики, поле show которых содержит значение True:



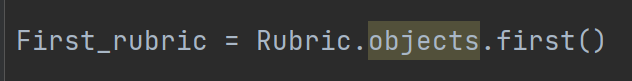
В качестве значения параметра также может быть использован экземпляр класса Q, задающий сложные критерии фильтрации.

Если параметр не указан, то список связываемых записей будет включать все записи первичной модели;

□ related name — имя атрибута записи первичной модели, предназначенного для доступа к связанным записям вторичной модели, в виде строки:

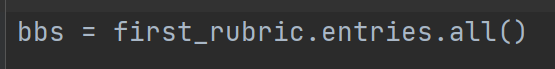


# Получаем первую рубрику



# Получаем доступ к связанным объявлениям через атрибут entries,

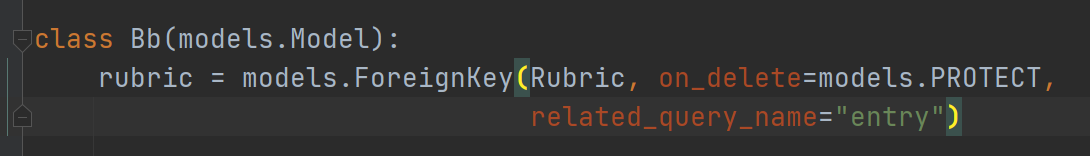
# указанный в параметре related\_name



Если доступ из записи первичной модели к связанным записям вторичной модели не требуется, можно указать Django не создавать такой атрибут и тем самым немного сэкономить системные ресурсы. Для этого достаточно присвоить параметру related\_name СИМВОЛ "ПЛЮС".

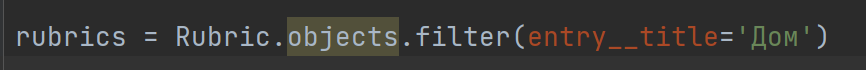
Если параметр не указан, то атрибут такого рода получит стандартное имя вида <имя связанной вторичной модели>\_Bb

□ reiated query name— имя фильтра, которое будет применяться во вторичной модели для фильтрации по значениям из записи первичной модели:



# Получаем все рубрики, содержащие объявления о продаже домов,

# воспользовавшись фильтром, заданным в параметре related\_query\_\_name



Если параметр не указан, то фильтр такого рода получит стандартное имя, совпадающее с именем класса вторичной модели.

□ to fieid— имя поля первичной модели, по которому будет выполнена связь, в виде строки. Такое поле должно быть помечено как уникальное (параметр Unique конструктора должен иметь значение True).

Если параметр не указан, связывание выполняется по ключевому полю первичной модели — неважно, созданному явно или неявно;

□ db constraint — если True, то в таблице базы данных будет создана связь, позволяющая сохранять ссылочную целостность; если False, ссылочная целостность будет поддерживаться только на уровне Django.

Значение по умолчанию — True. Менять его на False имеет смысл, только если модель создается на основе уже существующей базы с некорректными данными.

Связь "один-с-одним” соединяет одну запись первичной модели с одной записью вторичной модели. Может применяться для объединения моделей, одна из которых хранит данные, дополняющие данные из другой модели. Такая связь создается в классе вторичной модели объявлением поля типа Onetoonefieid. Вот формат конструктора этого класса:

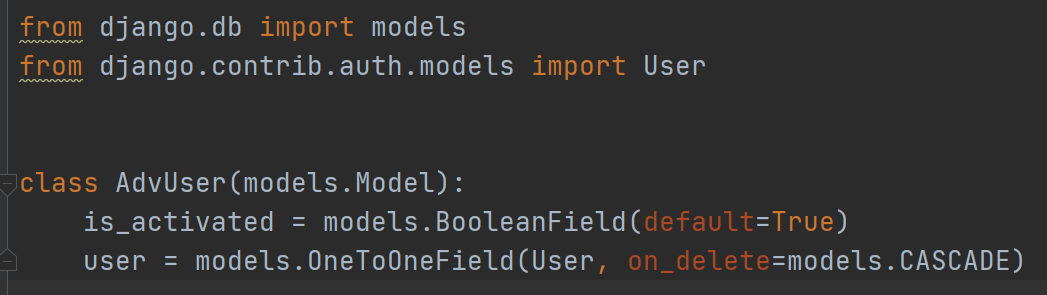
Onetoonefield(<связываемая первичная модель>,

On\_delete=<поведение при удалении записи>[,

<остальные параметры>] )

Первые два параметра точно такие же, как и у конструктора класса foreignfieid для хранения списка зарегистрированных на Django-сайте пользователей стандартная подсистема разграничения доступа использует особую модель. Ссылка на класс заменяемой модели пользователя хранится в параметре auth\_user\_model настроек проекта.

Давайте создадим модель advuser, хранящую дополнительные сведения о зарегистрированном пользователе. Свяжем ее со стандартной моделью пользователя user из модуля django.contrib.auth.models.



На уровне базы данных связь такого рода представляется точно таким же полем, что и связь типа "один-со-многими"

Конструктор класса поддерживает те же необязательные параметры, что и конструктор класса foreignfieid, плюс параметр parent link, применяемый при наследовании моделей.

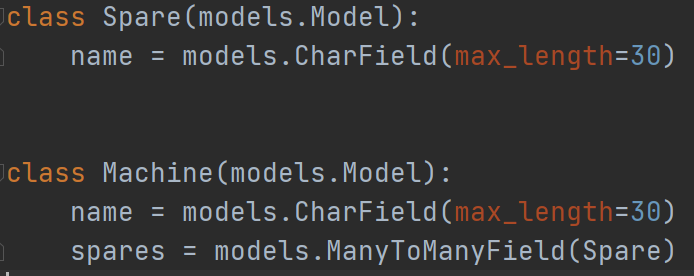
**Связь "многие-со-многими"**

Связь "многие-со-многими” соединяет произвольное число записей одной модели с произвольным числом записей другой (обе модели здесь выступают как равноправные, и определить, какая из них первичная, а какая вторичная, не представляется возможным). Для создания такой связи нужно объявить в одной из моделей (но не в обеих сразу!) Поле внешнего ключа типа manytomanyfieid. Вот формат его конструктора:

Manytomanyfield(<вторая связываемая модель>[, «остальные параметры>]}

Первый параметр задается в таком же формате, что и в конструкторах классов Foreignfieid и onetoonefield модель, в которой было объявлено поле внешнего ключа, назовем ведущей, а вторую модель — ведомой.

Для примера создадим модели Machine и spare, из которых первая, ведущая, будет хранить готовые машины, а вторая, ведомая, — отдельные детали для них.



В отличие от связей описанных ранее типов, имя поля, образующего связь ’’многие со-многими”, рекомендуется записывать во множественном числе. Что и логично — ведь такая связь позволяет связать произвольное число записей, что называется, с обеих сторон.

На уровне базы данных для представления связи такого типа создается таблица, по умолчанию имеющая ИМЯ вида <псевдоним приложения>\_<имя класса ведущей модели>\_<имя класса ведомой модели> (связующая таблица}. Она будет иметь ключевое поле Id и по одному полю с именем вида <имя класса связываемой модели>\_±& на каждую из связываемых моделей.

Конструктор класса manytomanyfieid поддерживает дополнительные необязательные параметры limit\_choices\_to, related\_name, related\_query\_name И db\_constraint, а также следующие:

□ symmetrical— используется только в тех случаях, когда модель связывается сама с собой. Если True, Django создаст симметричную связь, действующую в обоих направлениях (применительно к нашему случаю: если какая-то деталь А входит в машину Б, то машина Б содержит деталь А). Если False, то связь будет асимметричной (чисто гипотетически: Иванов любит колбасу, однако колбаса не любит Иванова). Значение по умолчанию — True.

Для асимметричной связи Django создаст в классе модели атрибут для доступа к записям связанной модели в обратном направлении □ through — класс модели, которая представляет связующую таблицу (связующая модель} либо в виде ссылки, либо в виде имени, представленном строкой. Если класс не указан, то связующая таблица будет создана самим Django. При использовании связующей модели нужно иметь в виду следующее:

• поле внешнего ключа для связи объявляется и в ведущей, и в ведомой моделях. При создании этих полей следует указать как саму связующую модель (параметр through), так и поля внешних ключей, по которым будет установлена связь (параметр through fieids, описанный далее);

• в связующей модели следует явно объявить поля внешних ключей для установления связи с обеими связываемыми моделями: и ведущей, и ведомой;

□ through fieids — используется, если связь устанавливается через связующую модель, записанную в параметре through конструктора. Указывает поля внешних ключей, по которым будет создаваться связь. Значение параметра должно представлять собой кортеж из двух элементов: имени поля ведущей модели и имени поля ведомой модели, записанных в виде строк. Если параметр не указан, то поля будут созданы самим фреймворком.

Пример использования связующей модели для установления связи "многие-со многими” и правильного заполнения параметров through и through\_fieids будет □ db tabie — имя связующей таблицы. Обычно применяется, если связующая модель не используется. Если оно не указано, то связующая таблица получит имя по умолчанию.

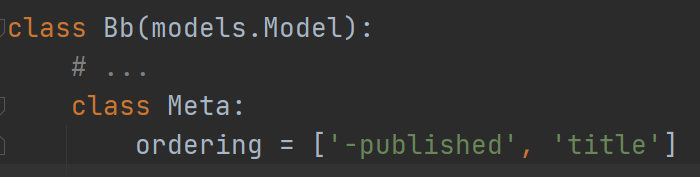
1. Параметры самой модели

Параметры самой модели описываются различными атрибутами класса Meta, вложенного в класс модели и не являющегося производным ни от какого класса. Вот список этих атрибутов:

□ verbose name — название сущности, хранящейся в модели, которое будет выводиться на экран. Если не указано, используется имя класса модели;

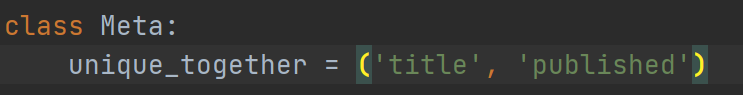
□ verbose name plural — название набора сущностей, хранящихся в модели, которая будет выводиться на экран. Если не указано, используется имя класса модели во множественном числе;

□ ordering— параметры сортировки записей модели по умолчанию. Задаются в виде последовательности имен полей, по которым должна выполняться сортировка, представленных строками. Если перед именем поля поставить символ ’’Минус”, то порядок сортировки будет обратным. Пример:



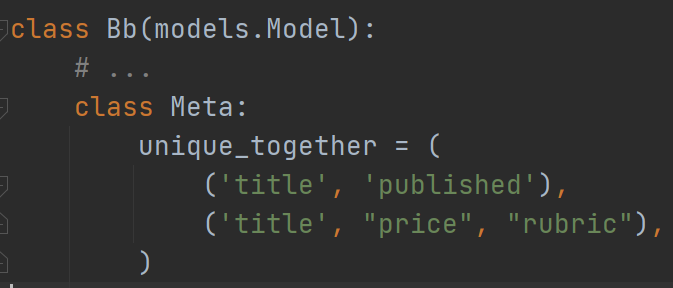
Сортируем записи модели сначала по убыванию значения поля published, а потом по возрастанию значения поля title;

□ unique together— последовательность имен полей, представленных в виде строк, которые должны хранить уникальные в пределах таблицы комбинации значений. При попытке занести в них уже имеющуюся в таблице комбинацию значений будет возбуждено исключение validationerror ИЗ модуля django.core, Exceptions. Пример:



Теперь комбинация названия товара и временной отметки публикации объявления должна быть уникальной в пределах модели. Добавить в тот же день еще одно объявление о продаже того же товара не получится.

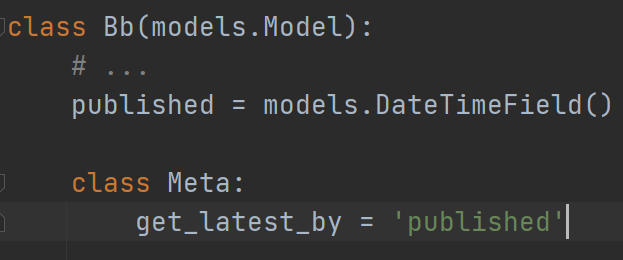
Можно указать несколько подобных групп полей, объединив их в последовательность:



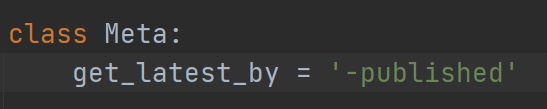
Теперь уникальными должны быть и комбинация названия товара и временной отметки публикации, и комбинация названия товара, цены и рубрики;

□ get latest by — ИМЯ ПОЛЯ типа datefield ИЛИ datetimefield, которое будет ВЗЯТО в расчет при получении наиболее поздней или наиболее ранней записи с помощью метода latest о или earliest о соответственно, вызванного без параметров. Можно задать:

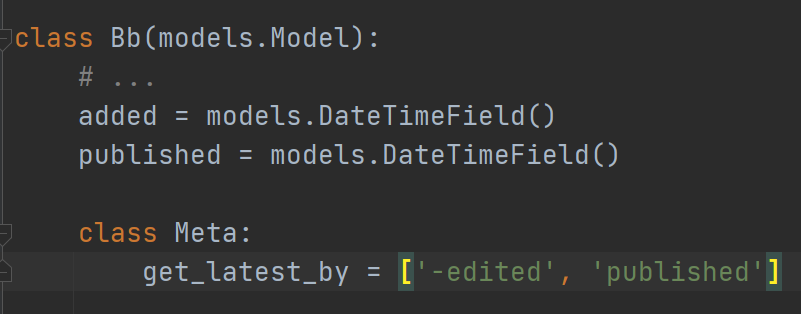
• имя поля в виде строки — тогда в расчет будет взято только это поле:



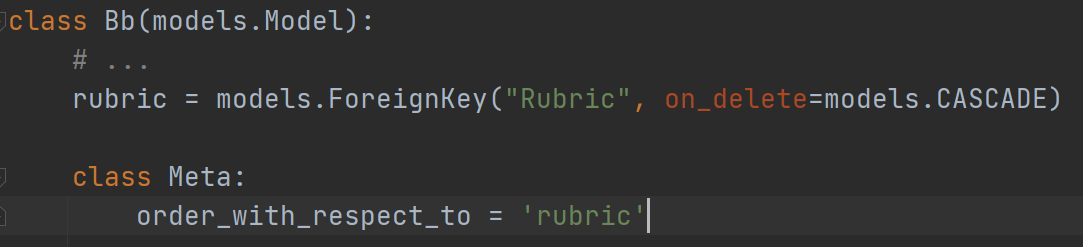
Теперь метод latest о вернет запись с наиболее поздним значением временной отметки, хранящемся в поле published. Если имя поля предварить символом ’’минус”, то порядок сортировки окажется обратным, и при вызове latest о мы получим, напротив, самую раннюю запись, а при вызове метода earliest () — самую позднюю:



• последовательность имен полей — тогда в расчет будут взяты значения всех этих полей, и, если у каких-то записей первое поле хранит одинаковые значения, будет проверяться значение второго поля и т. Д.:



□ order with respect to — позволяет сделать набор записей произвольно упорядочиваемым. В качестве значения параметра задается строка с именем поля текущей модели, и в дальнейшем записи, в которых это поле хранит одно и то же значение, могут быть упорядочены произвольным образом. Пример:



Теперь объявления, относящиеся к одной и той же рубрике, могут быть переупорядочены произвольным образом.

При указании в модели этого параметра в таблице будет дополнительно создано поле с именем вида <имя поля, заданного в качестве значения параметра>\_order.

Оно будет хранить целочисленное значение, указывающее порядковый номер текущей записи в последовательности. Одновременно вновь созданное поле с порядковым номером будет задано в качестве значения параметра ordering (см. Ранее). Следовательно, записи, которые мы извлечем из модели, по умолчанию будут отсортированы по значению этого поля. Указать другие параметры сортировки в таком случае будет невозможно;

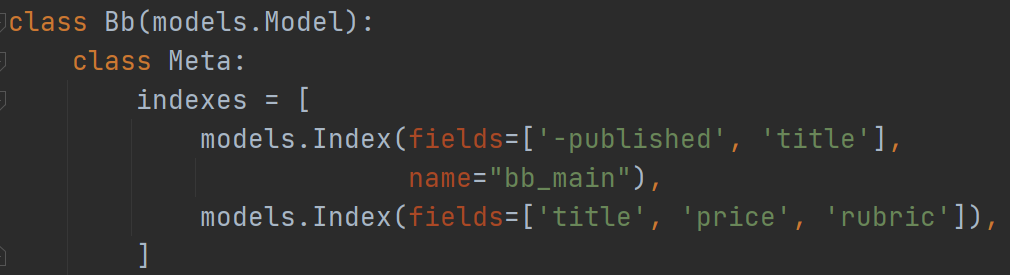
□ indexes — последовательность индексов, включающих в себя несколько полей.

Каждый элемент такой последовательности должен представлять собой экземпляр класса index из модуля django.db.models. Формат конструктора класса:

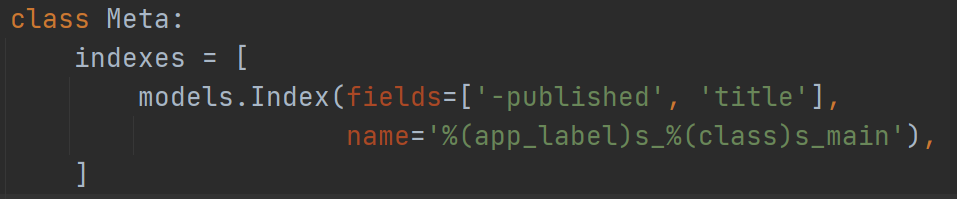
Index(fields=[][, name=None][, condition=None])

В параметре fields указывается список или кортеж строк с именами полей, которые должны быть включены в индекс. По умолчанию сортировка значений поля выполняется по их возрастанию, а чтобы отсортировать по убыванию, нужно предварить имя поля знаком "минус".

Параметр name задает имя индекса— если он не указан, то имя будет создано самим фреймворком. Пример:

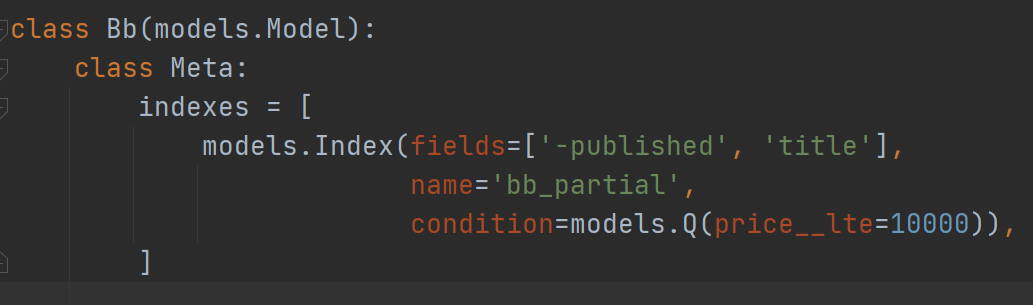


Начиная с Django 3.0, в имени индекса можно применять заменители % (app iabei) s и % (class) s, обозначающие соответственно псевдоним приложения и имя класса модели. Пример:



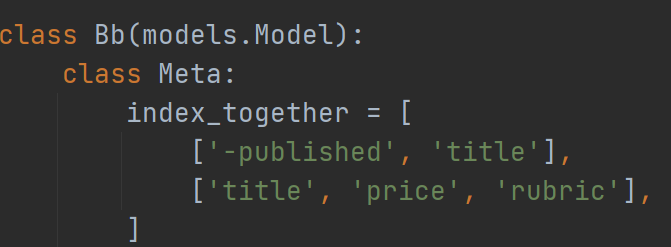
Параметр condition, поддерживаемый начиная c Django 2.2, задает критерий, которому должны удовлетворять записи, включаемые в индекс (mysql и Mariadb не поддерживают такие индексы). Критерий записывается с помощью экземпляров класса Q. Если он указан, также следует задать имя индекса в параметре name.

Пример создания индекса, включающего только товары с ценой менее 10 000:

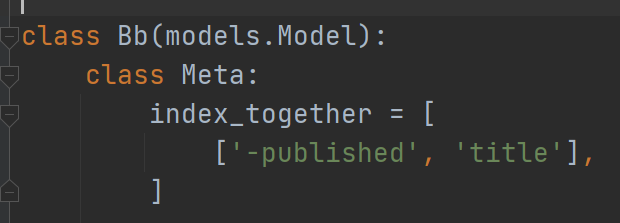


Mysql и mariadb не поддерживают подобного рода индексы, и параметр Condition в их случае игнорируется;

□ index together— предлагает другой способ создания индексов, содержащих несколько полей. Строки с именами полей указываются в виде последовательности, а набор таких последовательностей также объединяется в последовательность. Пример:



Если нужно создать всего один такой индекс, последовательность с именами его полей можно просто присвоить этому параметру:



□ defauit reiated name — имя атрибута записи первичной модели, предназначенного для доступа к связанным записям вторичной модели, в виде строки. Соответствует параметру reiated name конструкторов классов полей, предназначенных для установления связей между моделями. Неявно задает значения параметрам reiated name И related query name конструкторов;

□ db tabie —имя таблицы, в которой хранятся данные модели. Если оно не указано, ТО таблица получит ИМЯ вида <псевдоним приложения>\_<имя класса модели>.

Например, для модели вь приложения bboard в базе данных будет создана таблица bboard bb;

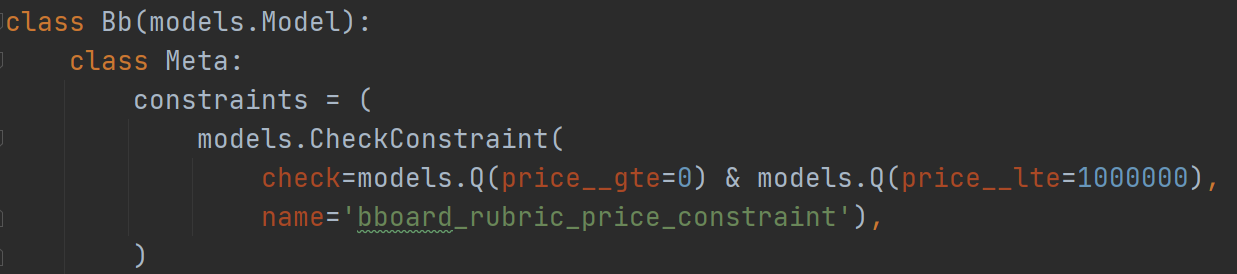
□ constraints (появился в Django 2.2) — условия, которым должны удовлетворять данные, заносимые в запись. В качестве значения указывается список или кортеж экземпляров классов, каждый из которых задает одно условие. Django включает два таких класса:

• checkconstraint — критерий, которому должны удовлетворять значения, заносимые в поля модели. Формат конструктора этого класса:

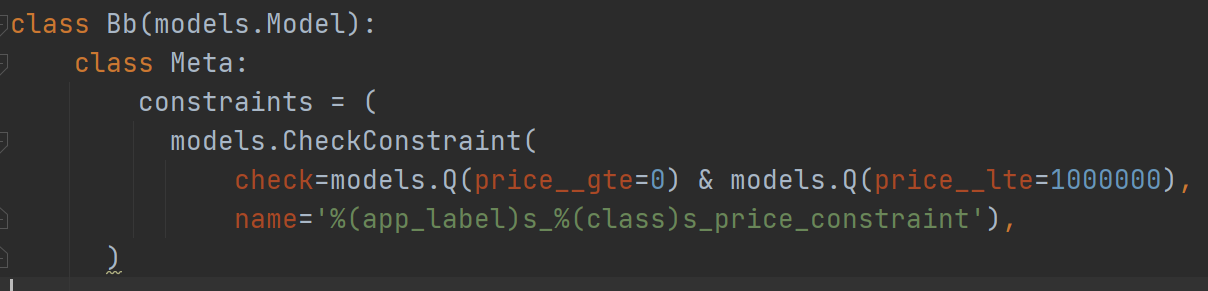
Checkconstraint(check=<критерии>, name=<имя условия>)

Критерий указывается экземплярами класса Q. Имя условия задается в виде строки и должно быть уникальным в пределах проекта.

Пример условия, требующего, чтобы задаваемая в объявлении цена находилась в диапазоне от 0 до 1 000 000:



Начиная c Django 3.0, в имени условия можно применять заменители % (app label) s и % (class) s, обозначающие соответственно псевдоним приложения и имя класса модели. Пример:



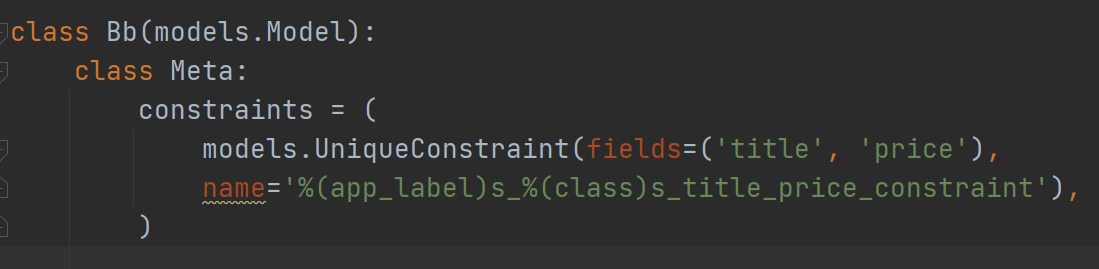
Если заносимые в запись значения не удовлетворяют заданным критериям, то при попытке сохранить запись будет возбуждено исключение integrityerror из модуля django.db, которое следует обработать программно. Никаких сообщений об ошибках ввода при этом на веб-страницу выведено не будет;

• uniqueconstraint — набор полей, которые должны хранить комбинации значений, уникальные в пределах таблицы. Конструктор класса вызывается в таком формате:

Uniqueconstraint(fields=<Ha6op полей>, пате=<имя условия>[,Condition=None])

Набор полей задается в виде списка или кортежа, содержащего строки с именами полей. Имя условия указывается в виде строки и должно быть уникальным в пределах проекта. Начиная с Django 3.0, в нем можно применять заменители % (app iabei)s и % (class) s, обозначающие соответственно псевдоним приложения и имя класса модели.

Пример условия, требующего уникальности комбинаций из названия товара и его цены:

****

Параметр condition задает дополнительный критерий, записанный с применением экземпляров класса q, Если он указан, то заданное условие будет применяться только к записям, удовлетворяющим этому критерию.

Пример условия, аналогичного приведенному ранее, но распространяющегося лишь на транспортные средства:



Поведение модели при нарушении заданного условия различается в зависимости от того, был ли указан параметр condition. Если он не был задан, на странице появится соответствующее сообщение об ошибке. В противном случае сообщение не появится, а будет возбуждено исключение Integrityerror из модуля django.db, которое следует обработать программно.

В параметре constraint может быть указано произвольное число условий, заданных разными классами. Условия, задаваемые в параметре constraints, обрабатываются на уровне СУБД (а не Django), для чего в базе данных создаются соответствующие правила.