**Поля отношений**[**¶**](https://django.fun/docs/django/ru/3.0/ref/models/fields/#module-django.db.models.fields.related)

Джанго также определяет набор полей, которые представляют отношения.

**ForeignKey[¶](https://django.fun/docs/django/ru/3.0/ref/models/fields/" \l "foreignkey" \o "Ссылка на этот заголовок)**

***class*ForeignKey(*to*, *on\_delete*, *\*\*options*)**[**[исходный код]**](https://django.fun/docs/django/ru/3.0/_modules/django/db/models/fields/related/#ForeignKey)[**¶**](https://django.fun/docs/django/ru/3.0/ref/models/fields/#django.db.models.ForeignKey)

Отношения многие-к-одному. Требуются два позиционных аргумента: класс, к которому относится модель, и опция **[on\_delete](https://django.fun/docs/django/ru/3.0/ref/models/fields/" \l "django.db.models.ForeignKey.on_delete" \o "django.db.models.ForeignKey.on_delete)**.

Чтобы создать рекурсивное отношение - объект, который имеет отношение «многие-к-одному» с самим собой - используйте **models.ForeignKey('self', on\_delete = models.CASCADE)**.

Если вам нужно создать отношение на модель, которая еще не была определена, вы можете использовать имя модели, а не сам объект модели:

**from** **django.db** **import** models

**class** **Car**(models.Model):

manufacturer = models.ForeignKey(

'Manufacturer',

on\_delete=models.CASCADE,

)

*# ...*

**class** **Manufacturer**(models.Model):

*# ...*

**pass**

Отношения, определенные следующим образом [абстрактные модели](https://django.fun/docs/django/ru/3.0/topics/db/models/#abstract-base-classes), разрешаются, когда модель подклассифицируется как конкретная модель, и не относятся к **app\_label** абстрактной модели:

products/models.py[**¶**](https://django.fun/docs/django/ru/3.0/ref/models/fields/#id3)

**from** **django.db** **import** models

**class** **AbstractCar**(models.Model):

manufacturer = models.ForeignKey('Manufacturer', on\_delete=models.CASCADE)

**class** **Meta**:

abstract = **True**

production/models.py[**¶**](https://django.fun/docs/django/ru/3.0/ref/models/fields/#id4)

**from** **django.db** **import** models

**from** **products.models** **import** AbstractCar

**class** **Manufacturer**(models.Model):

**pass**

**class** **Car**(AbstractCar):

**pass**

*# Car.manufacturer will point to `production.Manufacturer` here.*

Для ссылки на модели, определенные в другом приложении, вы можете явно указать модель с полной меткой приложения. Например, если описанная выше модель **Manufacturer** определена в другом приложении под названием **production**, вам необходимо использовать:

**class** **Car**(models.Model):

manufacturer = models.ForeignKey(

'production.Manufacturer',

on\_delete=models.CASCADE,

)

Такого рода ссылки, называемые ленивыми отношениями, могут быть полезны при разрешении циклических зависимостей импорта между двумя приложениями.

Индекс базы данных автоматически создается для **ForeignKey**. Вы можете отключить это, установив **[db\_index](https://django.fun/docs/django/ru/3.0/ref/models/fields/" \l "django.db.models.Field.db_index" \o "django.db.models.Field.db_index)** в **False**. Возможно, вы захотите избежать накладных расходов на индекс, если вы создаете внешний ключ для согласованности, а не для объединений, или если вы будете создавать альтернативный индекс, такой как индекс с частичным или множественным столбцом.

**Использование в базе данных**[**¶**](https://django.fun/docs/django/ru/3.0/ref/models/fields/#database-representation)

За кулисами Django добавляет **"\_id"** к имени поля, чтобы создать имя столбца базы данных. В приведенном выше примере таблица базы данных для модели **Car** будет иметь столбец **factory\_id**. (Вы можете изменить это явно, указав **[db\_column](https://django.fun/docs/django/ru/3.0/ref/models/fields/" \l "django.db.models.Field.db_column" \o "django.db.models.Field.db_column)**). Однако ваш код никогда не должен иметь дело с именем столбца базы данных, если вы не пишете пользовательский SQL. Вы всегда будете иметь дело с именами полей вашего модельного объекта.

**Аргументы**[**¶**](https://django.fun/docs/django/ru/3.0/ref/models/fields/#arguments)

[**ForeignKey**](https://django.fun/docs/django/ru/3.0/ref/models/fields/#django.db.models.ForeignKey) принимает другие аргументы, которые определяют детали работы отношений.

ForeignKey.on\_delete**[¶](https://django.fun/docs/django/ru/3.0/ref/models/fields/" \l "django.db.models.ForeignKey.on_delete" \o "Ссылка на это определение)**

При удалении объекта, на который ссылается **[ForeignKey](https://django.fun/docs/django/ru/3.0/ref/models/fields/" \l "django.db.models.ForeignKey" \o "django.db.models.ForeignKey)**, Django будет эмулировать поведение ограничения SQL, заданного аргументом **[on\_delete](https://django.fun/docs/django/ru/3.0/ref/models/fields/" \l "django.db.models.ForeignKey.on_delete" \o "django.db.models.ForeignKey.on_delete)**. Например, если у вас есть обнуляемым **[ForeignKey](https://django.fun/docs/django/ru/3.0/ref/models/fields/" \l "django.db.models.ForeignKey" \o "django.db.models.ForeignKey)** и вы хотите, чтобы он был установлен в null, когда ссылочный объект удален:

user = models.ForeignKey(

User,

models.SET\_NULL,

blank=**True**,

null=**True**,

)

**on\_delete** не создает ограничение SQL в базе данных. Поддержка опций каскадного уровня базы данных [может быть реализовано позже](https://code.djangoproject.com/ticket/21961).

Возможные значения для **[on\_delete](https://django.fun/docs/django/ru/3.0/ref/models/fields/" \l "django.db.models.ForeignKey.on_delete" \o "django.db.models.ForeignKey.on_delete)** находятся в **[django.db.models](https://django.fun/docs/django/ru/3.0/topics/db/models/" \l "module-django.db.models" \o "django.db.models)**:

* CASCADE[[исходный код]](https://django.fun/docs/django/ru/3.0/_modules/django/db/models/deletion/#CASCADE)[**¶**](https://django.fun/docs/django/ru/3.0/ref/models/fields/#django.db.models.CASCADE)

Каскадное удаление. Django эмулирует поведение ограничения SQL ON DELETE CASCADE, а также удаляет объект, содержащий ForeignKey.

[**Model.delete()**](https://django.fun/docs/django/ru/3.0/ref/models/instances/#django.db.models.Model.delete) не вызывается в связанных моделях, но сигналы **[pre\_delete](https://django.fun/docs/django/ru/3.0/ref/signals/" \l "django.db.models.signals.pre_delete" \o "django.db.models.signals.pre_delete)** и **[post\_delete](https://django.fun/docs/django/ru/3.0/ref/signals/" \l "django.db.models.signals.post_delete" \o "django.db.models.signals.post_delete)** отправляются для всех удаленных объектов.

* PROTECT[[исходный код]](https://django.fun/docs/django/ru/3.0/_modules/django/db/models/deletion/#PROTECT)[**¶**](https://django.fun/docs/django/ru/3.0/ref/models/fields/#django.db.models.PROTECT)

Предотвращает удаление объекта, на который есть ссылка, путем вызова **[ProtectedError](https://django.fun/docs/django/ru/3.0/ref/exceptions/" \l "django.db.models.ProtectedError" \o "django.db.models.ProtectedError)**, подкласса **[django.db.IntegrityError](https://django.fun/docs/django/ru/3.0/ref/exceptions/" \l "django.db.IntegrityError" \o "django.db.IntegrityError)**.

* SET\_NULL[[исходный код]](https://django.fun/docs/django/ru/3.0/_modules/django/db/models/deletion/#SET_NULL)[**¶**](https://django.fun/docs/django/ru/3.0/ref/models/fields/#django.db.models.SET_NULL)

Устанавливает **[ForeignKey](https://django.fun/docs/django/ru/3.0/ref/models/fields/" \l "django.db.models.ForeignKey" \o "django.db.models.ForeignKey)** null; это возможно только в том случае, если **[null](https://django.fun/docs/django/ru/3.0/ref/models/fields/" \l "django.db.models.Field.null" \o "django.db.models.Field.null)** равно **True**.

* SET\_DEFAULT[[исходный код]](https://django.fun/docs/django/ru/3.0/_modules/django/db/models/deletion/#SET_DEFAULT)[**¶**](https://django.fun/docs/django/ru/3.0/ref/models/fields/#django.db.models.SET_DEFAULT)

Устанавливает для **[ForeignKey](https://django.fun/docs/django/ru/3.0/ref/models/fields/" \l "django.db.models.ForeignKey" \o "django.db.models.ForeignKey)** значение по умолчанию; значение по умолчанию для **[ForeignKey](https://django.fun/docs/django/ru/3.0/ref/models/fields/" \l "django.db.models.ForeignKey" \o "django.db.models.ForeignKey)** должно быть указано в описании поля.

* SET()[[исходный код]](https://django.fun/docs/django/ru/3.0/_modules/django/db/models/deletion/#SET)[**¶**](https://django.fun/docs/django/ru/3.0/ref/models/fields/#django.db.models.SET)

Устанавливает для **[ForeignKey](https://django.fun/docs/django/ru/3.0/ref/models/fields/" \l "django.db.models.ForeignKey" \o "django.db.models.ForeignKey)** значение, переданное в [**SET()**](https://django.fun/docs/django/ru/3.0/ref/models/fields/#django.db.models.SET), или, если передан вызываемый объект, результат его вызова. В большинстве случаев передача вызываемого будет необходима, чтобы избежать выполнения запросов во время импорта вашего models.py:

**from** **django.conf** **import** settings

**from** **django.contrib.auth** **import** get\_user\_model

**from** **django.db** **import** models

**def** get\_sentinel\_user():

**return** get\_user\_model().objects.get\_or\_create(username='deleted')[0]

**class** **MyModel**(models.Model):

user = models.ForeignKey(

settings.AUTH\_USER\_MODEL,

on\_delete=models.SET(get\_sentinel\_user),

)

* DO\_NOTHING[[исходный код]](https://django.fun/docs/django/ru/3.0/_modules/django/db/models/deletion/#DO_NOTHING)[**¶**](https://django.fun/docs/django/ru/3.0/ref/models/fields/#django.db.models.DO_NOTHING)

Не предпринимает никаких действий. Если ваша база данных обеспечивает ссылочную целостность, это вызовет **[IntegrityError](https://django.fun/docs/django/ru/3.0/ref/exceptions/" \l "django.db.IntegrityError" \o "django.db.IntegrityError)**, если вы вручную не добавите ограничение SQL **ON DELETE** в поле базы данных.

ForeignKey.limit\_choices\_to**[¶](https://django.fun/docs/django/ru/3.0/ref/models/fields/" \l "django.db.models.ForeignKey.limit_choices_to" \o "Ссылка на это определение)**

Устанавливает ограничение на доступные варианты выбора для этого поля, когда это поле отображается с помощью **ModelForm** или в админке (по умолчанию все объекты в наборе запросов доступны для выбора). Можно использовать словарь, объект [**Q**](https://django.fun/docs/django/ru/3.0/ref/models/querysets/#django.db.models.Q) или вызываемый объект, возвращающий словарь или объект [**Q**](https://django.fun/docs/django/ru/3.0/ref/models/querysets/#django.db.models.Q).

Например:

staff\_member = models.ForeignKey(

User,

on\_delete=models.CASCADE,

limit\_choices\_to={'is\_staff': **True**},

)

заставляет соответствующее поле в **ModelForm** перечислять только **Users**, которые имеют **is\_staff=True**. Это может быть полезно в админке Django.

Вызываемая форма может быть полезна, например, при использовании совместно с модулем Python **datetime** для ограничения выбора по диапазону дат. Например:

**def** limit\_pub\_date\_choices():

**return** {'pub\_date\_\_lte': datetime.date.utcnow()}

limit\_choices\_to = limit\_pub\_date\_choices

Если **limit\_choices\_to** равно или возвращает [**Q объект**](https://django.fun/docs/django/ru/3.0/ref/models/querysets/#django.db.models.Q), который полезен для [сложных запросов](https://django.fun/docs/django/ru/3.0/topics/db/queries/#complex-lookups-with-q), тогда он будет влиять только на варианты, доступные в админке, если поле не указано в **[raw\_id\_fields](https://django.fun/docs/django/ru/3.0/ref/contrib/admin/" \l "django.contrib.admin.ModelAdmin.raw_id_fields" \o "django.contrib.admin.ModelAdmin.raw_id_fields)** в **ModelAdmin** для модели.

**Примечание**

Если для **limit\_choices\_to** используется вызываемый объект, он будет вызываться каждый раз, когда создается новая форма. Он также может быть вызван при проверке модели, например, командами управления или в админке. Админка создает наборы запросов для проверки входных данных своей формы в различных крайних случаях несколько раз, поэтому существует вероятность, что ваш вызываемый объект может быть вызван несколько раз.

ForeignKey.related\_name**[¶](https://django.fun/docs/django/ru/3.0/ref/models/fields/" \l "django.db.models.ForeignKey.related_name" \o "Ссылка на это определение)**

Имя, используемое для отношения от связанного объекта обратно к нему. Это также значение по умолчанию для **[related\_query\_name](https://django.fun/docs/django/ru/3.0/ref/models/fields/" \l "django.db.models.ForeignKey.related_query_name" \o "django.db.models.ForeignKey.related_query_name)** (имя, используемое для имени обратного фильтра из целевой модели). Смотрите [документацию по связанным объектам](https://django.fun/docs/django/ru/3.0/topics/db/queries/#backwards-related-objects) для полного объяснения и примера. Обратите внимание, что вы должны установить это значение при определении отношений [абстрактных моделей](https://django.fun/docs/django/ru/3.0/topics/db/models/#abstract-base-classes); и когда вы делаете это, то [доступен специальный синтаксис](https://django.fun/docs/django/ru/3.0/topics/db/models/#abstract-related-name).

Если вы предпочитаете, чтобы Django не создавал обратную связь, установите для **related\_name** значение **'+'** или завершите его с помощью **'+'**. Например, это гарантирует, что модель **User** не будет иметь обратной связи с этой моделью:

user = models.ForeignKey(

User,

on\_delete=models.CASCADE,

related\_name='+',

)

ForeignKey.related\_query\_name[**¶**](https://django.fun/docs/django/ru/3.0/ref/models/fields/#django.db.models.ForeignKey.related_query_name)

Имя, используемое для обратного имени фильтра из целевой модели. По умолчанию используется значение **[related\_name](https://django.fun/docs/django/ru/3.0/ref/models/fields/" \l "django.db.models.ForeignKey.related_name" \o "django.db.models.ForeignKey.related_name)** или **[default\_related\_name](https://django.fun/docs/django/ru/3.0/ref/models/options/" \l "django.db.models.Options.default_related_name" \o "django.db.models.Options.default_related_name)**, если установлено, в противном случае по умолчанию используется имя модели:

*# Declare the ForeignKey with related\_query\_name*

**class** **Tag**(models.Model):

article = models.ForeignKey(

Article,

on\_delete=models.CASCADE,

related\_name="tags",

related\_query\_name="tag",

)

name = models.CharField(max\_length=255)

*# That's now the name of the reverse filter*

Article.objects.filter(tag\_\_name="important")

Например **[related\_name](https://django.fun/docs/django/ru/3.0/ref/models/fields/" \l "django.db.models.ForeignKey.related_name" \o "django.db.models.ForeignKey.related_name)**, **related\_query\_name** поддерживает интерполяцию меток приложения и классов посредством [специального синтаксиса](https://django.fun/docs/django/ru/3.0/topics/db/models/#abstract-related-name).

ForeignKey.to\_field**[¶](https://django.fun/docs/django/ru/3.0/ref/models/fields/" \l "django.db.models.ForeignKey.to_field" \o "Ссылка на это определение)**

Поле связанного объекта, к которому относится отношение. По умолчанию Django использует первичный ключ связанного объекта. Если вы ссылаетесь на другое поле, это поле должно иметь **unique=True**.

ForeignKey.db\_constraint**[¶](https://django.fun/docs/django/ru/3.0/ref/models/fields/" \l "django.db.models.ForeignKey.db_constraint" \o "Ссылка на это определение)**

Определяет, следует ли создать ограничение в базе данных для этого внешнего ключа. По умолчанию используется **True**, и это почти наверняка то, что вы хотите; установка этого значения в **False** может быть очень плохой для целостности данных. Тем не менее, вот несколько сценариев, где вы можете сделать это:

* У вас есть устаревшие данные, которые не действительны.
* Вы разделяете свою базу данных.

Если для этого параметра установлено значение **False**, доступ к связанному объекту, который не существует, вызовет исключение **DoesNotExist**.

ForeignKey.swappable**[¶](https://django.fun/docs/django/ru/3.0/ref/models/fields/" \l "django.db.models.ForeignKey.swappable" \o "Ссылка на это определение)**

Управляет реакцией фреймворка миграций, если этот **[ForeignKey](https://django.fun/docs/django/ru/3.0/ref/models/fields/" \l "django.db.models.ForeignKey" \o "django.db.models.ForeignKey)** указывает на заменяемую модель. Если это **True** - по умолчанию - тогда, если **[ForeignKey](https://django.fun/docs/django/ru/3.0/ref/models/fields/" \l "django.db.models.ForeignKey" \o "django.db.models.ForeignKey)** указывает на модель, которая соответствует текущему значению **settings.AUTH\_USER\_MODEL** (или другой сменной настройке модели), связь будет хранится в миграции с использованием ссылки на настройку, а не на модель напрямую.

Вы можете изменить это значение на **False**, только если уверены, что ваша модель всегда должна указывать на замененную модель, например, если это модель профиля, разработанная специально для вашей пользовательской модели.

Установка этого значения в **False** не означает, что вы можете ссылаться на заменяемую модель, даже если она поменяна местами - **False** означает, что миграции, выполненные с помощью этого ForeignKey, всегда будут ссылаться на указанную вами точную модель (так что это не удастся легко сломать, если пользователь пытается запустить с моделью User, которую вы не поддерживаете, например).

Если сомневаетесь, оставьте для него значение по умолчанию **True**.

**ManyToManyField**[**¶**](https://django.fun/docs/django/ru/3.0/ref/models/fields/#manytomanyfield)

***class*ManyToManyField(*to*, *\*\*options*)**[**[исходный код]**](https://django.fun/docs/django/ru/3.0/_modules/django/db/models/fields/related/#ManyToManyField)[**¶**](https://django.fun/docs/django/ru/3.0/ref/models/fields/#django.db.models.ManyToManyField)

Отношения многие ко многим. Требуется позиционный аргумент: класс, к которому относится модель, который работает точно так же, как и для **[ForeignKey](https://django.fun/docs/django/ru/3.0/ref/models/fields/" \l "django.db.models.ForeignKey" \o "django.db.models.ForeignKey)**, включая [рекурсивные](https://django.fun/docs/django/ru/3.0/ref/models/fields/#recursive-relationships) и [ленивые](https://django.fun/docs/django/ru/3.0/ref/models/fields/#lazy-relationships) отношения.

Связанные объекты можно добавлять, удалять или создавать с помощью поля **[RelatedManager](https://django.fun/docs/django/ru/3.0/ref/models/relations/" \l "django.db.models.fields.related.RelatedManager" \o "django.db.models.fields.related.RelatedManager)**.

**Использование в базе данных**[**¶**](https://django.fun/docs/django/ru/3.0/ref/models/fields/#id1)

За кулисами Django создает промежуточную таблицу соединений для представления отношения «многие ко многим». По умолчанию это имя таблицы генерируется с использованием имени поля «многие ко многим» и имени таблицы для модели, в которой оно содержится. Поскольку некоторые базы данных не поддерживают имена таблиц выше определенной длины, эти имена таблиц будут автоматически усечены, и будет использован хеш уникальности, например, **Author\_books\_9cdf**. Вы можете вручную указать имя объединяемой таблицы с помощью параметра **[db\_table](https://django.fun/docs/django/ru/3.0/ref/models/fields/" \l "django.db.models.ManyToManyField.db_table" \o "django.db.models.ManyToManyField.db_table)**.

**Аргументы**[**¶**](https://django.fun/docs/django/ru/3.0/ref/models/fields/#manytomany-arguments)

[**ManyToManyField**](https://django.fun/docs/django/ru/3.0/ref/models/fields/#django.db.models.ManyToManyField) принимает дополнительный набор аргументов - все необязательные - которые управляют работой отношений.

ManyToManyField.related\_name**[¶](https://django.fun/docs/django/ru/3.0/ref/models/fields/" \l "django.db.models.ManyToManyField.related_name" \o "Ссылка на это определение)**

То же, что **[ForeignKey.related\_name](https://django.fun/docs/django/ru/3.0/ref/models/fields/" \l "django.db.models.ForeignKey.related_name" \o "django.db.models.ForeignKey.related_name)**.

ManyToManyField.related\_query\_name[**¶**](https://django.fun/docs/django/ru/3.0/ref/models/fields/#django.db.models.ManyToManyField.related_query_name)

То же, что **[ForeignKey.related\_query\_name](https://django.fun/docs/django/ru/3.0/ref/models/fields/" \l "django.db.models.ForeignKey.related_query_name" \o "django.db.models.ForeignKey.related_query_name)**.

ManyToManyField.limit\_choices\_to[**¶**](https://django.fun/docs/django/ru/3.0/ref/models/fields/#django.db.models.ManyToManyField.limit_choices_to)

То же, что **[ForeignKey.limit\_choices\_to](https://django.fun/docs/django/ru/3.0/ref/models/fields/" \l "django.db.models.ForeignKey.limit_choices_to" \o "django.db.models.ForeignKey.limit_choices_to)**.

**limit\_choices\_to** не влияет на использование **ManyToManyField** с пользовательской промежуточной таблицей, указанной с помощью параметра **[through](https://django.fun/docs/django/ru/3.0/ref/models/fields/" \l "django.db.models.ManyToManyField.through" \o "django.db.models.ManyToManyField.through)**.

ManyToManyField.symmetrical**[¶](https://django.fun/docs/django/ru/3.0/ref/models/fields/" \l "django.db.models.ManyToManyField.symmetrical" \o "Ссылка на это определение)**

Используется только в определении ManyToManyFields на самого себя. Рассмотрим следующую модель:

**from** **django.db** **import** models

**class** **Person**(models.Model):

friends = models.ManyToManyField("self")

Когда Django обрабатывает эту модель, он идентифицирует, что у него есть **[ManyToManyField](https://django.fun/docs/django/ru/3.0/ref/models/fields/" \l "django.db.models.ManyToManyField" \o "django.db.models.ManyToManyField)**, и в результате он не добавляет атрибут **person\_set** к классу **Person**. Вместо этого предполагается, что **[ManyToManyField](https://django.fun/docs/django/ru/3.0/ref/models/fields/" \l "django.db.models.ManyToManyField" \o "django.db.models.ManyToManyField)** является симметричным, то есть, если я ваш друг, то вы мой друг.

Если вам не нужна симметрия в отношениях «многие ко многим» с **self**, установите **[symmetrical](https://django.fun/docs/django/ru/3.0/ref/models/fields/" \l "django.db.models.ManyToManyField.symmetrical" \o "django.db.models.ManyToManyField.symmetrical)** в **False**. Это заставит Django добавить дескриптор для обратной связи, что сделает отношения **[ManyToManyField](https://django.fun/docs/django/ru/3.0/ref/models/fields/" \l "django.db.models.ManyToManyField" \o "django.db.models.ManyToManyField)** несимметричными.

Changed in Django 3.0:

Стало разрешено указывать **symmetrical=True** для рекурсивных отношений «многие ко многим» с использованием промежуточной модели.

ManyToManyField.through**[¶](https://django.fun/docs/django/ru/3.0/ref/models/fields/" \l "django.db.models.ManyToManyField.through" \o "Ссылка на это определение)**

Django автоматически создаст таблицу для управления отношениями «многие ко многим». Однако, если вы хотите указать промежуточную таблицу вручную, вы можете использовать опцию **[through](https://django.fun/docs/django/ru/3.0/ref/models/fields/" \l "django.db.models.ManyToManyField.through" \o "django.db.models.ManyToManyField.through)**, чтобы указать модель Django, представляющую промежуточную таблицу, которую вы хотите использовать.

Чаще всего этот параметр используется, когда вы хотите связать [дополнительные данные с отношением «многие ко многим»](https://django.fun/docs/django/ru/3.0/topics/db/models/#intermediary-manytomany).

**Примечание**

Рекурсивные отношения, использующие промежуточную модель и определяемые как симметричные (то есть с помощью **[symmetrical=False](https://django.fun/docs/django/ru/3.0/ref/models/fields/" \l "django.db.models.ManyToManyField.symmetrical" \o "django.db.models.ManyToManyField.symmetrical)**, которая используется по умолчанию), не могут определить имена обратных средств доступа, поскольку они будут одинаковыми. Вам необходимо установить **[related\_name](https://django.fun/docs/django/ru/3.0/ref/models/fields/" \l "django.db.models.ForeignKey.related_name" \o "django.db.models.ForeignKey.related_name)** хотя бы для одного из них. Если вы предпочитаете, чтобы Django не создавал обратную связь, установите для **related\_name** значение **'+'**.

Если вы не укажете явную модель **through**, все еще существует неявный класс модели **through**, который вы можете использовать для прямого доступа к таблице, созданной для хранения ассоциации. Он имеет три поля для связи моделей.

Если исходная и целевая модели различаются, создаются следующие поля:

* **id**: первичный ключ отношения.
* **<containing\_model>\_id**: **id** модели, которая объявляет **ManyToManyField**.
* **<other\_model>\_id**: `` id`` модели, на которую указывает **ManyToManyField**.

Если **ManyToManyField** указывает на одну и ту же модель, генерируются следующие поля:

* **id**: первичный ключ отношения.
* **from\_<model>\_id**: `` id`` экземпляра, который указывает на модель (то есть исходный экземпляр).
* **to\_<model>\_id**: `` id`` экземпляра, на который указывает отношение (то есть экземпляр целевой модели).

Этот класс можно использовать для запроса связанных записей для данного экземпляра модели, например, нормальной модели.

ManyToManyField.through\_fields**[¶](https://django.fun/docs/django/ru/3.0/ref/models/fields/" \l "django.db.models.ManyToManyField.through_fields" \o "Ссылка на это определение)**

Используется только когда указана пользовательская промежуточная модель. Django обычно определяет, какие поля промежуточной модели использовать для автоматического установления отношения «многие ко многим». Однако рассмотрим следующие модели:

**from** **django.db** **import** models

**class** **Person**(models.Model):

name = models.CharField(max\_length=50)

**class** **Group**(models.Model):

name = models.CharField(max\_length=128)

members = models.ManyToManyField(

Person,

through='Membership',

through\_fields=('group', 'person'),

)

**class** **Membership**(models.Model):

group = models.ForeignKey(Group, on\_delete=models.CASCADE)

person = models.ForeignKey(Person, on\_delete=models.CASCADE)

inviter = models.ForeignKey(

Person,

on\_delete=models.CASCADE,

related\_name="membership\_invites",

)

invite\_reason = models.CharField(max\_length=64)

**Membership** имеет *два* внешних ключа для **Person** (**person** и **inviter**), что делает отношения неоднозначными, и Django не может знать, какой из них использовать. В этом случае вы должны явно указать, какие внешние ключи должен использовать Django, используя **through\_fields**, как в примере выше.

**through\_fields** принимает 2-х значный кортеж **('field1', 'field2')**, где **field1** - имя внешнего ключа модели, для которого определен класс **[ManyToManyField](https://django.fun/docs/django/ru/3.0/ref/models/fields/" \l "django.db.models.ManyToManyField" \o "django.db.models.ManyToManyField)** (**group** в данном случае) и **field2** - имя внешнего ключа для целевой модели (в данном случае **person**).

Если у вас есть более одного внешнего ключа в промежуточной модели для любой (или даже обеих) моделей, участвующих в отношении многие-ко-многим, вы *должны* указать **through\_fields**. Это также относится к [рекурсивным отношениям](https://django.fun/docs/django/ru/3.0/ref/models/fields/#recursive-relationships), когда используется модель-посредник и существует более двух внешних ключей для модели, или вы хотите явно указать, какие два Django следует использовать.

ManyToManyField.db\_table**[¶](https://django.fun/docs/django/ru/3.0/ref/models/fields/" \l "django.db.models.ManyToManyField.db_table" \o "Ссылка на это определение)**

Имя таблицы, создаваемой для хранения данных «многие ко многим». Если это не предусмотрено, Django примет имя по умолчанию на основе имен: таблицы для модели, определяющей отношение, и имени самого поля.

ManyToManyField.db\_constraint**[¶](https://django.fun/docs/django/ru/3.0/ref/models/fields/" \l "django.db.models.ManyToManyField.db_constraint" \o "Ссылка на это определение)**

Определяет, следует ли создавать ограничения в базе данных для внешних ключей в промежуточной таблице. По умолчанию используется **True**, и это почти наверняка то, что вы хотите; установка этого значения в **False** может быть очень плохой для целостности данных. Тем не менее, вот несколько сценариев, где вы можете сделать это:

* У вас есть устаревшие данные, которые не действительны.
* Вы разделяете свою базу данных.

Будет ошибкой пропускать оба параметра: **db\_constraint** и **through**.

ManyToManyField.swappable**[¶](https://django.fun/docs/django/ru/3.0/ref/models/fields/" \l "django.db.models.ManyToManyField.swappable" \o "Ссылка на это определение)**

Управляет реакцией фреймворка миграции, если этот **[ManyToManyField](https://django.fun/docs/django/ru/3.0/ref/models/fields/" \l "django.db.models.ManyToManyField" \o "django.db.models.ManyToManyField)** указывает на заменяемую модель. Если **True** - по умолчанию - тогда, если **[ManyToManyField](https://django.fun/docs/django/ru/3.0/ref/models/fields/" \l "django.db.models.ManyToManyField" \o "django.db.models.ManyToManyField)** указывает на модель, которая соответствует текущему значению **settings.AUTH\_USER\_MODEL** (или другой сменной настройки модели), отношение будет хранится в миграции с использованием ссылки на настройку, а не на модель напрямую.

Вы можете изменить это значение на **False**, только если уверены, что ваша модель всегда должна указывать на замененную модель, например, если это модель профиля, разработанная специально для вашей пользовательской модели.

Если сомневаетесь, оставьте для него значение по умолчанию **True**.

[**ManyToManyField**](https://django.fun/docs/django/ru/3.0/ref/models/fields/#django.db.models.ManyToManyField) не поддерживает [**validators**](https://django.fun/docs/django/ru/3.0/ref/models/fields/#django.db.models.Field.validators).

[**null**](https://django.fun/docs/django/ru/3.0/ref/models/fields/#django.db.models.Field.null) не имеет никакого эффекта, поскольку нет способа требовать отношений на уровне базы данных.

**OneToOneField[¶](https://django.fun/docs/django/ru/3.0/ref/models/fields/" \l "onetoonefield" \o "Ссылка на этот заголовок)**

***class*OneToOneField(*to*, *on\_delete*, *parent\_link=False*, *\*\*options*)**[**[исходный код]**](https://django.fun/docs/django/ru/3.0/_modules/django/db/models/fields/related/#OneToOneField)[**¶**](https://django.fun/docs/django/ru/3.0/ref/models/fields/#django.db.models.OneToOneField)

Отношения один-к-одному. Концептуально это похоже на **[ForeignKey](https://django.fun/docs/django/ru/3.0/ref/models/fields/" \l "django.db.models.ForeignKey" \o "django.db.models.ForeignKey)** с **[unique=True](https://django.fun/docs/django/ru/3.0/ref/models/fields/" \l "django.db.models.Field.unique" \o "django.db.models.Field.unique)**, но «обратная» сторона отношения будет напрямую возвращать один объект.

Это наиболее полезно в качестве первичного ключа модели, которая каким-то образом «расширяет» другую модель; [Мультитабличное наследование](https://django.fun/docs/django/ru/3.0/topics/db/models/" \l "multi-table-inheritance) реализуется путем добавления неявного отношения один-к-одному из дочерней модели, например, в родительскую модель.

Требуется один позиционный аргумент: класс, с которым будет связана модель. Это работает точно так же, как и для **[ForeignKey](https://django.fun/docs/django/ru/3.0/ref/models/fields/" \l "django.db.models.ForeignKey" \o "django.db.models.ForeignKey)**, включая все параметры, относящиеся к [рекурсивным](https://django.fun/docs/django/ru/3.0/ref/models/fields/#recursive-relationships) и [ленивым](https://django.fun/docs/django/ru/3.0/ref/models/fields/#lazy-relationships) отношениям.

Если вы не укажете аргумент **[related\_name](https://django.fun/docs/django/ru/3.0/ref/models/fields/" \l "django.db.models.ForeignKey.related_name" \o "django.db.models.ForeignKey.related_name)** для **OneToOneField**, Django будет использовать имя текущей модели в нижнем регистре в качестве значения по умолчанию.

Со следующим примером:

**from** **django.conf** **import** settings

**from** **django.db** **import** models

**class** **MySpecialUser**(models.Model):

user = models.OneToOneField(

settings.AUTH\_USER\_MODEL,

on\_delete=models.CASCADE,

)

supervisor = models.OneToOneField(

settings.AUTH\_USER\_MODEL,

on\_delete=models.CASCADE,

related\_name='supervisor\_of',

)

ваша итоговая модель **User** будет иметь следующие атрибуты:

**>>>** user = User.objects.get(pk=1)

**>>>** hasattr(user, 'myspecialuser')

True

**>>>** hasattr(user, 'supervisor\_of')

True

Исключение **DoesNotExist** возникает при обращении к обратному отношению, если запись в связанной таблице не существует. Например, если у пользователя нет супервизора, назначенного **MySpecialUser**:

**>>>** user.supervisor\_of

Traceback (most recent call last):

...

DoesNotExist: User matching query does not exist.

Кроме того, **OneToOneField** принимает все дополнительные аргументы, принятые **[ForeignKey](https://django.fun/docs/django/ru/3.0/ref/models/fields/" \l "django.db.models.ForeignKey" \o "django.db.models.ForeignKey)**, плюс один дополнительный аргумент:

OneToOneField.parent\_link**[¶](https://django.fun/docs/django/ru/3.0/ref/models/fields/" \l "django.db.models.OneToOneField.parent_link" \o "Ссылка на это определение)**

Когда **True** и используется в модели, которая наследуется от другого [concrete model](https://django.fun/docs/django/ru/3.0/glossary/" \l "term-concrete-model), указывает, что это поле следует использовать как ссылку на родительский класс, а не как дополнительное **OneToOneField**, которое обычно неявно создается путем создания подклассов.

Смотрите [Отношения один-к-одному](https://django.fun/docs/django/ru/3.0/topics/db/examples/one_to_one/) для примеров использования **OneToOneField**.

**Справочник API полей**[**¶**](https://django.fun/docs/django/ru/3.0/ref/models/fields/#field-api-reference)

***class*Field[[исходный код]](https://django.fun/docs/django/ru/3.0/_modules/django/db/models/fields/" \l "Field)**[**¶**](https://django.fun/docs/django/ru/3.0/ref/models/fields/#django.db.models.Field)

**Field - абстрактный класс, представляющий столбец таблицы базы данных. Django использует поля для создания таблицы базы данных ([db\_type()](https://django.fun/docs/django/ru/3.0/ref/models/fields/" \l "django.db.models.Field.db_type" \o "django.db.models.Field.db_type)), для отображения типов Python в базу данных ([get\_prep\_value()](https://django.fun/docs/django/ru/3.0/ref/models/fields/" \l "django.db.models.Field.get_prep_value" \o "django.db.models.Field.get_prep_value)) и наоборот ([from\_db\_value()](https://django.fun/docs/django/ru/3.0/ref/models/fields/" \l "django.db.models.Field.from_db_value" \o "django.db.models.Field.from_db_value)).**

**Таким образом, поле является фундаментальной частью в различных API Django, а именно**[**модели**](https://django.fun/docs/django/ru/3.0/ref/models/instances/#django.db.models.Model)**и**[**наборы запросов**](https://django.fun/docs/django/ru/3.0/ref/models/querysets/#django.db.models.query.QuerySet)**.**

**В моделях поле создается как атрибут класса и представляет определенный столбец таблицы, см.**[**Модели**](https://django.fun/docs/django/ru/3.0/topics/db/models/)**. Он имеет такие атрибуты, как [null](https://django.fun/docs/django/ru/3.0/ref/models/fields/" \l "django.db.models.Field.null" \o "django.db.models.Field.null) и [unique](https://django.fun/docs/django/ru/3.0/ref/models/fields/" \l "django.db.models.Field.unique" \o "django.db.models.Field.unique), а также методы, которые Django использует для сопоставления значения поля со значениями, специфичными для базы данных.**

**Field является подклассом [RegisterLookupMixin](https://django.fun/docs/django/ru/3.0/ref/models/lookups/" \l "django.db.models.lookups.RegisterLookupMixin" \o "django.db.models.lookups.RegisterLookupMixin) и, следовательно, и [Transform](https://django.fun/docs/django/ru/3.0/ref/models/lookups/" \l "django.db.models.Transform" \o "django.db.models.Transform) и [Lookup](https://django.fun/docs/django/ru/3.0/ref/models/lookups/" \l "django.db.models.Lookup" \o "django.db.models.Lookup) может быть зарегистрирован для использования в QuerySet (например, field\_name\_\_exact="foo"). Все**[**встроенные фильтры**](https://django.fun/docs/django/ru/3.0/ref/models/querysets/#field-lookups)**зарегистрированы по умолчанию.**

**Все встроенные поля Django, такие как [CharField](https://django.fun/docs/django/ru/3.0/ref/models/fields/" \l "django.db.models.CharField" \o "django.db.models.CharField), являются конкретными реализациями Field. Если вам нужно настраиваемое поле, вы можете создать подкласс любого из встроенных полей или написать Field с нуля. В любом случае смотрите [Writing custom model fields](https://django.fun/docs/django/ru/3.0/howto/custom-model-fields/).**

**description[¶](https://django.fun/docs/django/ru/3.0/ref/models/fields/" \l "django.db.models.Field.description" \o "Ссылка на это определение)**

**Подробное описание поля, например, для приложения [django.contrib.admindocs](https://django.fun/docs/django/ru/3.0/ref/contrib/admin/admindocs/" \l "module-django.contrib.admindocs" \o "django.contrib.admindocs: Django's admin documentation generator.).**

**Описание может иметь вид:**

**description = \_("String (up to %(max\_length)s)")**

**где аргументы интерполируются из поля \_\_dict\_\_.**

**descriptor\_class[¶](https://django.fun/docs/django/ru/3.0/ref/models/fields/" \l "django.db.models.Field.descriptor_class" \o "Ссылка на это определение)**

**New in Django 3.0.**

**Класс, реализующий**[**протокол дескриптора**](https://docs.python.org/3/reference/datamodel.html#descriptors)**, который создается и присваивается атрибуту экземпляра модели. Конструктор должен принимать один аргумент, экземпляр Field. Переопределение этого атрибута класса позволяет настроить поведение get и set.**

**Чтобы сопоставить Field с типом, специфичным для базы данных, Django предлагает несколько методов:**

**get\_internal\_type()**[**[исходный код]**](https://django.fun/docs/django/ru/3.0/_modules/django/db/models/fields/#Field.get_internal_type)[**¶**](https://django.fun/docs/django/ru/3.0/ref/models/fields/#django.db.models.Field.get_internal_type)

**Возвращает строку с именем этого поля для конкретных целей бэкэнда. По умолчанию возвращает имя класса.**

**Смотрите [Emulating built-in field types](https://django.fun/docs/django/ru/3.0/howto/custom-model-fields/" \l "emulating-built-in-field-types) для использования в пользовательских полях.**

**db\_type(*connection*)**[**[исходный код]**](https://django.fun/docs/django/ru/3.0/_modules/django/db/models/fields/#Field.db_type)[**¶**](https://django.fun/docs/django/ru/3.0/ref/models/fields/#django.db.models.Field.db_type)

**Возвращает тип данных столбца базы данных для [Field](https://django.fun/docs/django/ru/3.0/ref/models/fields/" \l "django.db.models.Field" \o "django.db.models.Field) с учетом connection.**

**Смотрите [Custom database types](https://django.fun/docs/django/ru/3.0/howto/custom-model-fields/" \l "custom-database-types) для использования в пользовательских полях.**

**rel\_db\_type(*connection*)**[**[исходный код]**](https://django.fun/docs/django/ru/3.0/_modules/django/db/models/fields/#Field.rel_db_type)[**¶**](https://django.fun/docs/django/ru/3.0/ref/models/fields/#django.db.models.Field.rel_db_type)

**Возвращает тип данных столбца базы данных для таких полей, как ForeignKey и OneToOneField, которые указывают на [Field](https://django.fun/docs/django/ru/3.0/ref/models/fields/" \l "django.db.models.Field" \o "django.db.models.Field), с учетом connection.**

**Смотрите [Custom database types](https://django.fun/docs/django/ru/3.0/howto/custom-model-fields/" \l "custom-database-types) для использования в пользовательских полях.**

**Существует три основных ситуации, в которых Django должен взаимодействовать с базой данных и полями базы данных:**

* **когда он запрашивает базу данных (значение Python -> значение базы данных)**
* **когда он загружает данные из базы данных (значение базы данных -> значение Python)**
* **когда он сохраняется в базе данных (значение Python -> значение базы данных)**

**При запросах используются [get\_db\_prep\_value()](https://django.fun/docs/django/ru/3.0/ref/models/fields/" \l "django.db.models.Field.get_db_prep_value" \o "django.db.models.Field.get_db_prep_value) и [get\_prep\_value()](https://django.fun/docs/django/ru/3.0/ref/models/fields/" \l "django.db.models.Field.get_prep_value" \o "django.db.models.Field.get_prep_value):**

**get\_prep\_value(*value*)**[**[исходный код]**](https://django.fun/docs/django/ru/3.0/_modules/django/db/models/fields/#Field.get_prep_value)[**¶**](https://django.fun/docs/django/ru/3.0/ref/models/fields/#django.db.models.Field.get_prep_value)

**value - это текущее значение атрибута модели, и метод должен возвращать данные в формате, подготовленном для использования в качестве параметра в запросе.**

**См.**[**Converting Python objects to query values**](https://django.fun/docs/django/ru/3.0/howto/custom-model-fields/#converting-python-objects-to-query-values)**для использования.**

**get\_db\_prep\_value(*value*, *connection*, *prepared=False*)**[**[исходный код]**](https://django.fun/docs/django/ru/3.0/_modules/django/db/models/fields/#Field.get_db_prep_value)[**¶**](https://django.fun/docs/django/ru/3.0/ref/models/fields/#django.db.models.Field.get_db_prep_value)

**Преобразует value в специфичное для бэкенда значение. По умолчанию возвращается value если prepare=True и**[**get\_prep\_value()**](https://django.fun/docs/django/ru/3.0/ref/models/fields/#django.db.models.Field.get_prep_value)**если False.**

**См.**[**Converting query values to database values**](https://django.fun/docs/django/ru/3.0/howto/custom-model-fields/#converting-query-values-to-database-values)**для использования.**

**При загрузке данных используется [from\_db\_value()](https://django.fun/docs/django/ru/3.0/ref/models/fields/" \l "django.db.models.Field.from_db_value" \o "django.db.models.Field.from_db_value):**

**from\_db\_value(*value*, *expression*, *connection*)**[**¶**](https://django.fun/docs/django/ru/3.0/ref/models/fields/#django.db.models.Field.from_db_value)

**Преобразует значение, возвращаемое базой данных, в объект Python. Это обратная сторона [get\_prep\_value()](https://django.fun/docs/django/ru/3.0/ref/models/fields/" \l "django.db.models.Field.get_prep_value" \o "django.db.models.Field.get_prep_value).**

**Этот метод не используется для большинства встроенных полей, так как серверная часть базы данных уже возвращает правильный тип Python, или сама серверная часть выполняет преобразование.**

**Смотрите**[**Converting values to Python objects**](https://django.fun/docs/django/ru/3.0/howto/custom-model-fields/#converting-values-to-python-objects)**для использования.**

**Примечание**

**По соображениям производительности from\_db\_value не реализовано как запрет на поля, которые не требуют этого (все поля Django). Следовательно, вы не можете вызвать super в вашем определении.**

**При сохранении используются [pre\_save()](https://django.fun/docs/django/ru/3.0/ref/models/fields/" \l "django.db.models.Field.pre_save" \o "django.db.models.Field.pre_save) и [get\_db\_prep\_save()](https://django.fun/docs/django/ru/3.0/ref/models/fields/" \l "django.db.models.Field.get_db_prep_save" \o "django.db.models.Field.get_db_prep_save):**

**get\_db\_prep\_save(*value*, *connection*)**[**[исходный код]**](https://django.fun/docs/django/ru/3.0/_modules/django/db/models/fields/#Field.get_db_prep_save)[**¶**](https://django.fun/docs/django/ru/3.0/ref/models/fields/#django.db.models.Field.get_db_prep_save)

**То же, что [get\_db\_prep\_value()](https://django.fun/docs/django/ru/3.0/ref/models/fields/" \l "django.db.models.Field.get_db_prep_value" \o "django.db.models.Field.get_db_prep_value), но вызывается, когда значение поля должно быть *сохранено* в базе данных. По умолчанию возвращает [get\_db\_prep\_value()](https://django.fun/docs/django/ru/3.0/ref/models/fields/" \l "django.db.models.Field.get_db_prep_value" \o "django.db.models.Field.get_db_prep_value).**

**pre\_save(*model\_instance*, *add*)**[**[исходный код]**](https://django.fun/docs/django/ru/3.0/_modules/django/db/models/fields/#Field.pre_save)[**¶**](https://django.fun/docs/django/ru/3.0/ref/models/fields/#django.db.models.Field.pre_save)

**Метод, вызываемый до [get\_db\_prep\_save()](https://django.fun/docs/django/ru/3.0/ref/models/fields/" \l "django.db.models.Field.get_db_prep_save" \o "django.db.models.Field.get_db_prep_save), чтобы подготовить значение перед сохранением (например, для [DateField.auto\_now](https://django.fun/docs/django/ru/3.0/ref/models/fields/" \l "django.db.models.DateField.auto_now" \o "django.db.models.DateField.auto_now)).**

**model\_instance - это экземпляр, которому принадлежит это поле, а add указывает, сохраняется ли экземпляр в базу данных в первый раз.**

**Он должен вернуть значение соответствующего атрибута из model\_instance для этого поля. Имя атрибута находится в self.attname (это настраивается с помощью [Field](https://django.fun/docs/django/ru/3.0/ref/models/fields/" \l "django.db.models.Field" \o "django.db.models.Field)).**

**Смотрите [Preprocessing values before saving](https://django.fun/docs/django/ru/3.0/howto/custom-model-fields/" \l "preprocessing-values-before-saving) для использования.**

**Поля часто получают свои значения в виде другого типа, либо из сериализации, либо из форм.**

**to\_python(*value*)**[**[исходный код]**](https://django.fun/docs/django/ru/3.0/_modules/django/db/models/fields/#Field.to_python)[**¶**](https://django.fun/docs/django/ru/3.0/ref/models/fields/#django.db.models.Field.to_python)

**Преобразует значение в правильный объект Python. Он действует как противоположность [value\_to\_string()](https://django.fun/docs/django/ru/3.0/ref/models/fields/" \l "django.db.models.Field.value_to_string" \o "django.db.models.Field.value_to_string) и также вызывается в [clean()](https://django.fun/docs/django/ru/3.0/ref/models/instances/" \l "django.db.models.Model.clean" \o "django.db.models.Model.clean).**

**Смотрите**[**Converting values to Python objects**](https://django.fun/docs/django/ru/3.0/howto/custom-model-fields/#converting-values-to-python-objects)**для использования.**

**Помимо сохранения в базе данных, поле также должно знать, как сериализовать его значение:**

**value\_from\_object(*obj*)**[**[исходный код]**](https://django.fun/docs/django/ru/3.0/_modules/django/db/models/fields/#Field.value_from_object)[**¶**](https://django.fun/docs/django/ru/3.0/ref/models/fields/#django.db.models.Field.value_from_object)

**Возвращает значение поля для данного экземпляра модели.**

**Этот метод часто используется методом [value\_to\_string()](https://django.fun/docs/django/ru/3.0/ref/models/fields/" \l "django.db.models.Field.value_to_string" \o "django.db.models.Field.value_to_string).**

**value\_to\_string(*obj*)**[**[исходный код]**](https://django.fun/docs/django/ru/3.0/_modules/django/db/models/fields/#Field.value_to_string)[**¶**](https://django.fun/docs/django/ru/3.0/ref/models/fields/#django.db.models.Field.value_to_string)

**Преобразует obj в строку. Используется для сериализации значения поля.**

**Смотрите**[**Converting field data for serialization**](https://django.fun/docs/django/ru/3.0/howto/custom-model-fields/#converting-model-field-to-serialization)**для использования.**

**При использовании**[**модельных форм**](https://django.fun/docs/django/ru/3.0/topics/forms/modelforms/#django.forms.ModelForm)**, Field должно знать, в каком поле формы оно должно быть представлено:**

**formfield(*form\_class=None*, *choices\_form\_class=None*, *\*\*kwargs*)**[**[исходный код]**](https://django.fun/docs/django/ru/3.0/_modules/django/db/models/fields/#Field.formfield)[**¶**](https://django.fun/docs/django/ru/3.0/ref/models/fields/#django.db.models.Field.formfield)

**Возвращает значение по умолчанию [django.forms.Field](https://django.fun/docs/django/ru/3.0/ref/forms/fields/" \l "django.forms.Field" \o "django.forms.Field) этого поля для [ModelForm](https://django.fun/docs/django/ru/3.0/topics/forms/modelforms/" \l "django.forms.ModelForm" \o "django.forms.ModelForm).**

**По умолчанию, если оба form\_class и choices\_form\_class являются None, он использует [CharField](https://django.fun/docs/django/ru/3.0/ref/forms/fields/" \l "django.forms.CharField" \o "django.forms.CharField). Если в поле есть [choices](https://django.fun/docs/django/ru/3.0/ref/models/fields/" \l "django.db.models.Field.choices" \o "django.db.models.Field.choices) и choices\_form\_class не указано, оно использует [TypedChoiceField](https://django.fun/docs/django/ru/3.0/ref/forms/fields/" \l "django.forms.TypedChoiceField" \o "django.forms.TypedChoiceField).**

**Смотрите**[**Specifying the form field for a model field**](https://django.fun/docs/django/ru/3.0/howto/custom-model-fields/#specifying-form-field-for-model-field)**для использования.**

**deconstruct()**[**[исходный код]**](https://django.fun/docs/django/ru/3.0/_modules/django/db/models/fields/#Field.deconstruct)[**¶**](https://django.fun/docs/django/ru/3.0/ref/models/fields/#django.db.models.Field.deconstruct)

**Возвращает кортеж из 4-х значений с достаточным количеством информации для воссоздания поля:**

1. **Название поля в модели.**
2. **Путь импорта поля (например, "django.db.models.IntegerField"). Это должна быть самая портативная версия, поэтому менее конкретная может быть лучше.**
3. **Список позиционных аргументов.**
4. **Словарь ключевых аргументов.**

**Этот метод должен быть добавлен в поля до версии 1.7 для переноса его данных с помощью [Migrations](https://django.fun/docs/django/ru/3.0/topics/migrations/).**

**Справочник атрибутов поля**[**¶**](https://django.fun/docs/django/ru/3.0/ref/models/fields/#field-attribute-reference)

Каждый экземпляр **Field** содержит несколько атрибутов, которые позволяют проанализировать его поведение. Используйте эти атрибуты вместо проверок **isinstance**, когда вам нужно написать код, который зависит от функциональности поля. Эти атрибуты могут использоваться вместе с [Model.\_meta API](https://django.fun/docs/django/ru/3.0/ref/models/meta/" \l "model-meta-field-api), чтобы сузить поиск для определенных типов полей. Пользовательские поля модели должны реализовывать эти флаги.

**Атрибуты для полей**[**¶**](https://django.fun/docs/django/ru/3.0/ref/models/fields/#attributes-for-fields)

Field.auto\_created**[¶](https://django.fun/docs/django/ru/3.0/ref/models/fields/" \l "django.db.models.Field.auto_created" \o "Ссылка на это определение)**

Логический флаг, который указывает, было ли поле создано автоматически, например, **OneToOneField**, используемый наследованием модели.

Field.concrete**[¶](https://django.fun/docs/django/ru/3.0/ref/models/fields/" \l "django.db.models.Field.concrete" \o "Ссылка на это определение)**

Логический флаг, который указывает, имеет ли поле столбец в базе данных, связанный с ним.

Field.hidden**[¶](https://django.fun/docs/django/ru/3.0/ref/models/fields/" \l "django.db.models.Field.hidden" \o "Ссылка на это определение)**

Логический флаг, который указывает, используется ли поле для поддержки функциональности другого не скрытого поля (например, поля **content\_type** и **object\_id**, которые составляют **GenericForeignKey**). Флаг **hidden** используется для того, чтобы отличить то, что составляет общедоступное подмножество полей модели от всех полей модели.

**Примечание**

[**Options.get\_fields ()**](https://django.fun/docs/django/ru/3.0/ref/models/meta/#django.db.models.options.Options.get_fields) исключает скрытые поля по умолчанию. Передайте **include\_hidden=True**, чтобы вернуть скрытые поля в результаты.

Field.is\_relation**[¶](https://django.fun/docs/django/ru/3.0/ref/models/fields/" \l "django.db.models.Field.is_relation" \o "Ссылка на это определение)**

Логический флаг, который указывает, содержит ли поле ссылки на одну или несколько других моделей для его функциональности (например, **ForeignKey**, **ManyToManyField**, **OneToOneField** и т.д.).

Field.model**[¶](https://django.fun/docs/django/ru/3.0/ref/models/fields/" \l "django.db.models.Field.model" \o "Ссылка на это определение)**

Возвращает модель, для которой определено поле. Если поле определено в суперклассе модели, **model** будет ссылаться на суперкласс, а не на класс экземпляра.

**Атрибуты для полей с отношениями**[**¶**](https://django.fun/docs/django/ru/3.0/ref/models/fields/#attributes-for-fields-with-relations)

Эти атрибуты используются для запроса количества элементов и других деталей отношения. Эти атрибуты присутствуют во всех полях; однако они будут иметь логические значения (а не **None**), если поле имеет тип отношения (**[Field.is\_relation=True](https://django.fun/docs/django/ru/3.0/ref/models/fields/" \l "django.db.models.Field.is_relation" \o "django.db.models.Field.is_relation)**).

Field.many\_to\_many**[¶](https://django.fun/docs/django/ru/3.0/ref/models/fields/" \l "django.db.models.Field.many_to_many" \o "Ссылка на это определение)**

Логический флаг, который устанавливается в **True**, если поле имеет отношение «многие ко многим»; иначе **False**. Единственное поле, включенное в Django, где **True**, это **ManyToManyField**.

Field.many\_to\_one**[¶](https://django.fun/docs/django/ru/3.0/ref/models/fields/" \l "django.db.models.Field.many_to_one" \o "Ссылка на это определение)**

Логический флаг, устанавливаемый в **True**, если поле имеет отношение многие-к-одному, например, **ForeignKey**; иначе``False``.

Field.one\_to\_many**[¶](https://django.fun/docs/django/ru/3.0/ref/models/fields/" \l "django.db.models.Field.one_to_many" \o "Ссылка на это определение)**

Логический флаг, устанавливаемый в **True**, если поле имеет отношение один ко многим, такое как **GenericRelation** или обратное для **ForeignKey**; иначе **False**.

Field.one\_to\_one**[¶](https://django.fun/docs/django/ru/3.0/ref/models/fields/" \l "django.db.models.Field.one_to_one" \o "Ссылка на это определение)**

Логический флаг, устанавливаемый в **True**, если поле имеет взаимно-однозначное отношение, например **OneToOneField**; иначе``False``.

Field.related\_model**[¶](https://django.fun/docs/django/ru/3.0/ref/models/fields/" \l "django.db.models.Field.related_model" \o "Ссылка на это определение)**

Указывает на модель, к которой относится поле. Например, **Author** в **ForeignKey(Author, on\_delete=models.CASCADE)**. **Related\_model** для **GenericForeignKey** всегда **None**.