
django tutorial

Выпуск 1.0

alekseyf

мар. 28, 2025

Contents:

Содержание модулей Python	9
Алфавитный указатель	11

Add your content using `reStructuredText` syntax. See the [reStructuredText](#) documentation for details.

```
class polls.models.User(id, name, email, phone, date_registered)
```

Базовые классы: `Model`

Параметры

- `id` (*BigAutoField*) – Primary key: ID
- `name` (*CharField*) – Name
- `email` (*EmailField*) – Email
- `phone` (*CharField*) – Phone
- `date_registered` (*DateTimeField*) – Date registered

Reverse relationships:

Параметры

`order` (Reverse `ForeignKey` from *Order*) – All orders of this user (related name of *user*)

`name`

Type: `CharField`

Name

A wrapper for a deferred-loading field. When the value is read from this

`email`

Type: `EmailField`

Email

A wrapper for a deferred-loading field. When the value is read from this

`phone`

Type: `CharField`

Phone

A wrapper for a deferred-loading field. When the value is read from this

`date_registered`

Type: `DateTimeField`

Date registered

A wrapper for a deferred-loading field. When the value is read from this

exception `DoesNotExist`

Базовые классы: `ObjectDoesNotExist`

exception `MultipleObjectsReturned`

Базовые классы: `MultipleObjectsReturned`

```
get_next_by_date_registered(*, field=<django.db.models.DateTimeField: date_registered>,
                           is_next=True, **kwargs)
```

Finds next instance based on *date_registered*. See `get_next_by_FOO()` for more information.

```
get_previous_by_date_registered(*, field=<django.db.models.DateTimeField: date_registered>,
                               is_next=False, **kwargs)
```

Finds previous instance based on *date_registered*. See `get_previous_by_FOO()` for more information.

`id`

Type: `BigAutoField`

Primary key: ID

A wrapper for a deferred-loading field. When the value is read from this

`objects = <django.db.models.Manager object>`

`order_set`

Type: Reverse `ForeignKey` from `Order`

All orders of this user (related name of `user`)

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager

`class polls.models.Product(id, name, category, price, stock_quantity, description)`

Базовые классы: `Model`

Параметры

- `id` (`BigAutoField`) – Primary key: ID
- `name` (`CharField`) – Name
- `category` (`CharField`) – Category
- `price` (`DecimalField`) – Price
- `stock_quantity` (`IntegerField`) – Stock quantity
- `description` (`TextField`) – Description

Reverse relationships:

Параметры

`orderitem` (Reverse `ForeignKey` from `OrderItem`) – All order items of this product (related name of `product`)

`name`

Type: `CharField`

Name

A wrapper for a deferred-loading field. When the value is read from this

`category`

Type: `CharField`

Category

A wrapper for a deferred-loading field. When the value is read from this

price

Type: `DecimalField`

Price

A wrapper for a deferred-loading field. When the value is read from this

stock_quantity

Type: `IntegerField`

Stock quantity

A wrapper for a deferred-loading field. When the value is read from this

description

Type: `TextField`

Description

A wrapper for a deferred-loading field. When the value is read from this

exception `DoesNotExist`

Базовые классы: `ObjectDoesNotExist`

exception `MultipleObjectsReturned`

Базовые классы: `MultipleObjectsReturned`

id

Type: `BigAutoField`

Primary key: ID

A wrapper for a deferred-loading field. When the value is read from this

objects = <django.db.models.Manager object>

orderitem_set

Type: Reverse `ForeignKey` from `OrderItem`

All order items of this product (related name of `product`)

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager

```
class polls.models.Order(id, user, date_ordered, status)
```

Базовые классы: `Model`

Параметры

- `id` (`BigAutoField`) – Primary key: ID
- `date_ordered` (`DateTimeField`) – Date ordered
- `status` (`CharField`) – Status

Relationship fields:

Параметры

`user` (`ForeignKey` to `User`) – User (related name: `order`)

Reverse relationships:

Параметры

`items` (Reverse `ForeignKey` from `OrderItem`) – All items of this order (related name of `order`)

`STATUS_CHOICES` = [('new', 'New'), ('processing', 'Processing'), ('shipped', 'Shipped'), ('completed', 'Completed')]

`user`

Type: `ForeignKey` to `User`

User (related name: `order`)

Accessor to the related object on the forward side of a many-to-one or one-to-one (via `ForwardOneToOneDescriptor` subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`date_ordered`

Type: `DateTimeField`

Date ordered

A wrapper for a deferred-loading field. When the value is read from this

`status`

Type: `CharField`

Status

Choices:

- new
- processing
- shipped
- completed

A wrapper for a deferred-loading field. When the value is read from this

exception `DoesNotExist`

Базовые классы: `ObjectDoesNotExist`

exception `MultipleObjectsReturned`

Базовые классы: `MultipleObjectsReturned`

`get_next_by_date_ordered(*, field=<django.db.models.DateTimeField: date_ordered>, is_next=True, **kwargs)`

Finds next instance based on `date_ordered`. See `get_next_by_F00()` for more information.

`get_previous_by_date_ordered(*, field=<django.db.models.DateTimeField: date_ordered>, is_next=False, **kwargs)`

Finds previous instance based on `date_ordered`. See `get_previous_by_F00()` for more information.

`get_status_display(*, field=<django.db.models.CharField: status>)`

Shows the label of the *status*. See `get_FOO_display()` for more information.

`id`

Type: `BigAutoField`

Primary key: ID

A wrapper for a deferred-loading field. When the value is read from this

`items`

Type: Reverse `ForeignKey` from *OrderItem*

All items of this order (related name of *order*)

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager

`objects = <django.db.models.Manager object>`

`user_id`

Internal field, use *user* instead.

`class polls.models.OrderItem(id, order, product, quantity)`

Базовые классы: `Model`

Параметры

- `id` (`BigAutoField`) – Primary key: ID
- `quantity` (`PositiveIntegerField`) – Quantity

Relationship fields:

Параметры

- `order` (`ForeignKey` to *Order*) – Order (related name: *items*)
- `product` (`ForeignKey` to *Product*) – Product (related name: *orderitem*)

`order`

Type: `ForeignKey` to *Order*

Order (related name: *items*)

Accessor to the related object on the forward side of a many-to-one or one-to-one (via `ForwardOneToOneDescriptor` subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

product

Type: `ForeignKey` to *Product*

Product (related name: *orderitem*)

Accessor to the related object on the forward side of a many-to-one or one-to-one (via `ForwardOneToOneDescriptor` subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

quantity

Type: `PositiveIntegerField`

Quantity

A wrapper for a deferred-loading field. When the value is read from this

exception `DoesNotExist`

Базовые классы: `ObjectDoesNotExist`

exception `MultipleObjectsReturned`

Базовые классы: `MultipleObjectsReturned`

id

Type: `BigAutoField`

Primary key: ID

A wrapper for a deferred-loading field. When the value is read from this

`objects = <django.db.models.Manager object>`

order_id

Internal field, use *order* instead.

product_id

Internal field, use *product* instead.

```
class polls.views.UserViewSet(**kwargs)
```

Базовые классы: `ModelViewSet`

```
queryset = QuerySet
```

```
serializer_class
```

псевдоним для `UserSerializer`

```
class polls.views.ProductViewSet(**kwargs)
```

Базовые классы: `ModelViewSet`

```
queryset = QuerySet
```

```
serializer_class
```

псевдоним для `ProductSerializer`

```
class polls.views.OrderViewSet(**kwargs)
```

Базовые классы: `ModelViewSet`

```

    queryset = QuerySet

    serializer_class
        псевдоним для OrderSerializer
class polls.views.OrderItemViewSet(**kwargs)
    Базовые классы: ModelViewSet

    queryset = QuerySet

    serializer_class
        псевдоним для OrderItemSerializer
polls.views.api_home(request, *args, **kwargs)
    Возвращает список всех доступных маршрутов API во всем проекте.
polls.views.collect_routes(patterns, request, prefix='')
    Рекурсивно собирает все маршруты API во всем Django-проекте и корректно формирует URL.
polls.views.get_base_url(request=None)
    Возвращает базовый URL в зависимости от окружения

```


p
polls.models, 1
polls.views, 6

A

`api_home()` (в модуле `polls.views`), 7

C

`category` (*ампубум* `polls.models.Product`), 2

`collect_routes()` (в модуле `polls.views`), 7

D

`date_ordered` (*ампубум* `polls.models.Order`), 4

`date_registered` (*ампубум* `polls.models.User`), 1

`description` (*ампубум* `polls.models.Product`), 3

E

`email` (*ампубум* `polls.models.User`), 1

G

`get_base_url()` (в модуле `polls.views`), 7

`get_next_by_date_ordered()` (*memod*
`polls.models.Order`), 4

`get_next_by_date_registered()` (*memod*
`polls.models.User`), 1

`get_previous_by_date_ordered()` (*memod*
`polls.models.Order`), 4

`get_previous_by_date_registered()` (*memod*
`polls.models.User`), 1

`get_status_display()` (*memod*
`polls.models.Order`), 4

I

`id` (*ампубум* `polls.models.Order`), 5

`id` (*ампубум* `polls.models.OrderItem`), 6

`id` (*ампубум* `polls.models.Product`), 3

`id` (*ампубум* `polls.models.User`), 1

`items` (*ампубум* `polls.models.Order`), 5

M

`module`

`polls.models`, 1

`polls.views`, 6

N

`name` (*ампубум* `polls.models.Product`), 2

`name` (*ампубум* `polls.models.User`), 1

O

`objects` (*ампубум* `polls.models.Order`), 5

`objects` (*ампубум* `polls.models.OrderItem`), 6

`objects` (*ампубум* `polls.models.Product`), 3

`objects` (*ампубум* `polls.models.User`), 2

`order` (*ампубум* `polls.models.OrderItem`), 5

`Order` (класс в `polls.models`), 3

`Order.DoesNotExist`, 4

`Order.MultipleObjectsReturned`, 4

`order_id` (*ампубум* `polls.models.OrderItem`), 6

`order_set` (*ампубум* `polls.models.User`), 2

`OrderItem` (класс в `polls.models`), 5

`OrderItem.DoesNotExist`, 6

`OrderItem.MultipleObjectsReturned`, 6

`orderitem_set` (*ампубум* `polls.models.Product`), 3

`OrderItemViewSet` (класс в `polls.views`), 7

`OrderViewSet` (класс в `polls.views`), 6

P

`phone` (*ампубум* `polls.models.User`), 1

`polls.models`

`module`, 1

`polls.views`

`module`, 6

`price` (*ампубум* `polls.models.Product`), 2

`product` (*ампубум* `polls.models.OrderItem`), 5

`Product` (класс в `polls.models`), 2

`Product.DoesNotExist`, 3

`Product.MultipleObjectsReturned`, 3

`product_id` (*ампубум* `polls.models.OrderItem`), 6

`ProductViewSet` (класс в `polls.views`), 6

Q

`quantity` (*ампубум* `polls.models.OrderItem`), 6

[queryset \(ампубум polls.views.OrderItemViewSet\), 7](#)
[queryset \(ампубум polls.views.OrderViewSet\), 6](#)
[queryset \(ампубум polls.views.ProductViewSet\), 6](#)
[queryset \(ампубум polls.views.UserViewSet\), 6](#)

S

[serializer_class \(ампубум polls.views.OrderItemViewSet\), 7](#)
[serializer_class \(ампубум polls.views.OrderViewSet\), 7](#)
[serializer_class \(ампубум polls.views.ProductViewSet\), 6](#)
[serializer_class \(ампубум polls.views.UserViewSet\), 6](#)
[status \(ампубум polls.models.Order\), 4](#)
[STATUS_CHOICES \(ампубум polls.models.Order\), 4](#)
[stock_quantity \(ампубум polls.models.Product\), 3](#)

U

[user \(ампубум polls.models.Order\), 4](#)
[User \(класс в polls.models\), 1](#)
[User.DoesNotExist, 1](#)
[User.MultipleObjectsReturned, 1](#)
[user_id \(ампубум polls.models.Order\), 5](#)
[UserViewSet \(класс в polls.views\), 6](#)