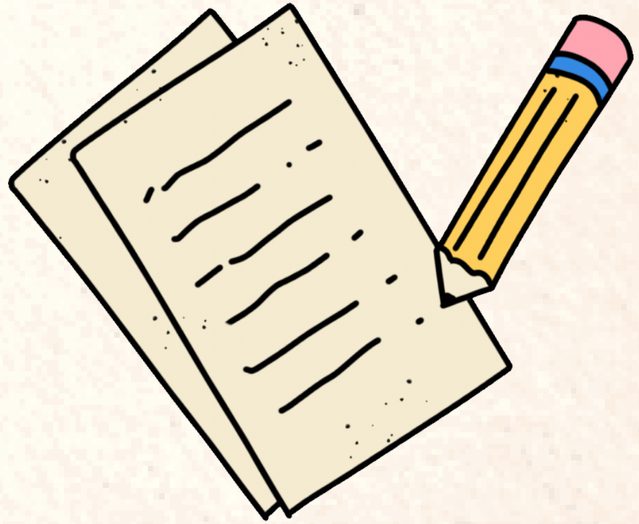

Hi!

OUTLINES

- Relationships in Django Model

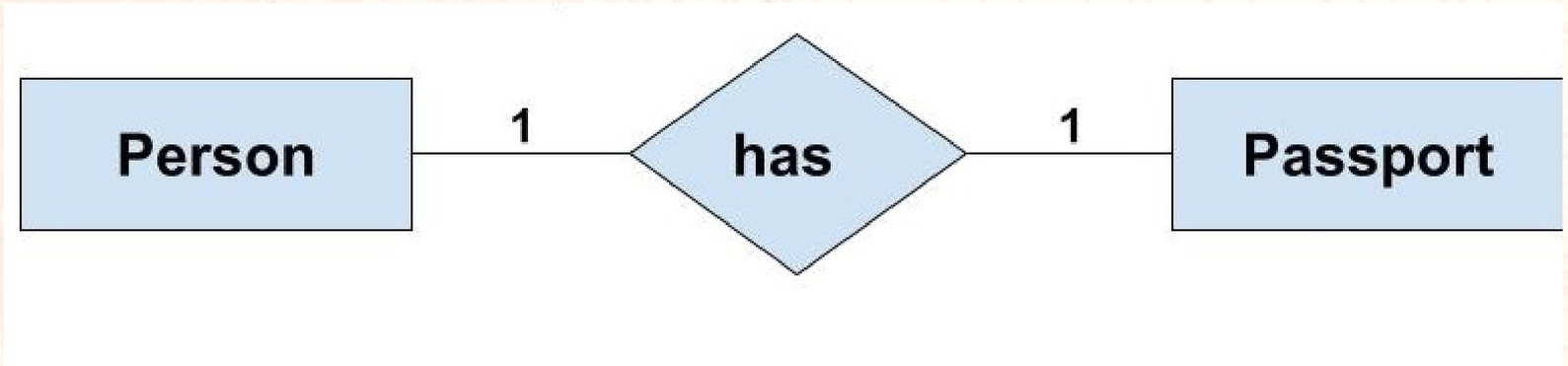


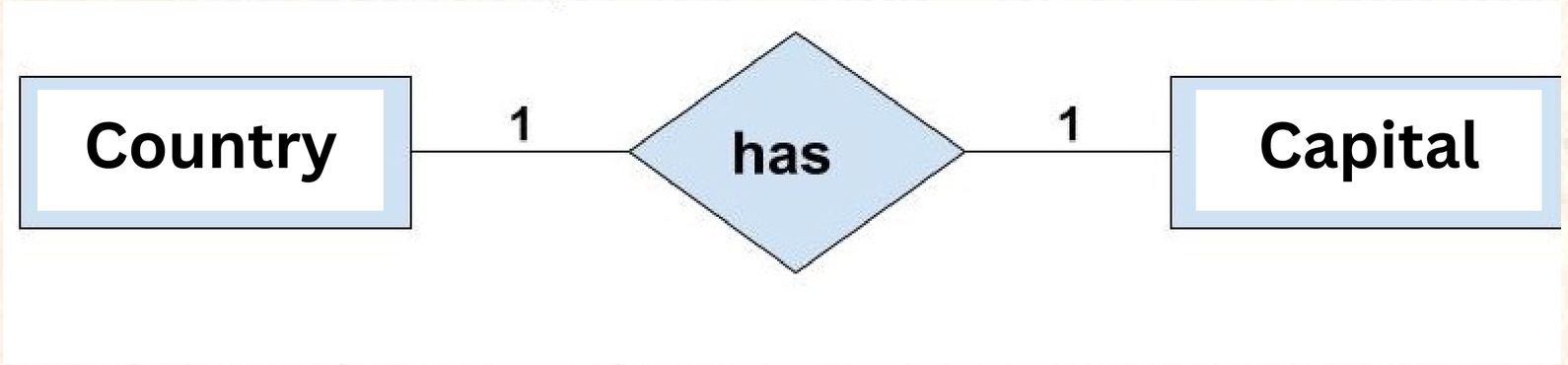
RELATIONSHIP IN DJANGO MODEL

Django offers three types of database relationships

- One to One Relationship
- Many to One Relationship
- Many to Many Relationships

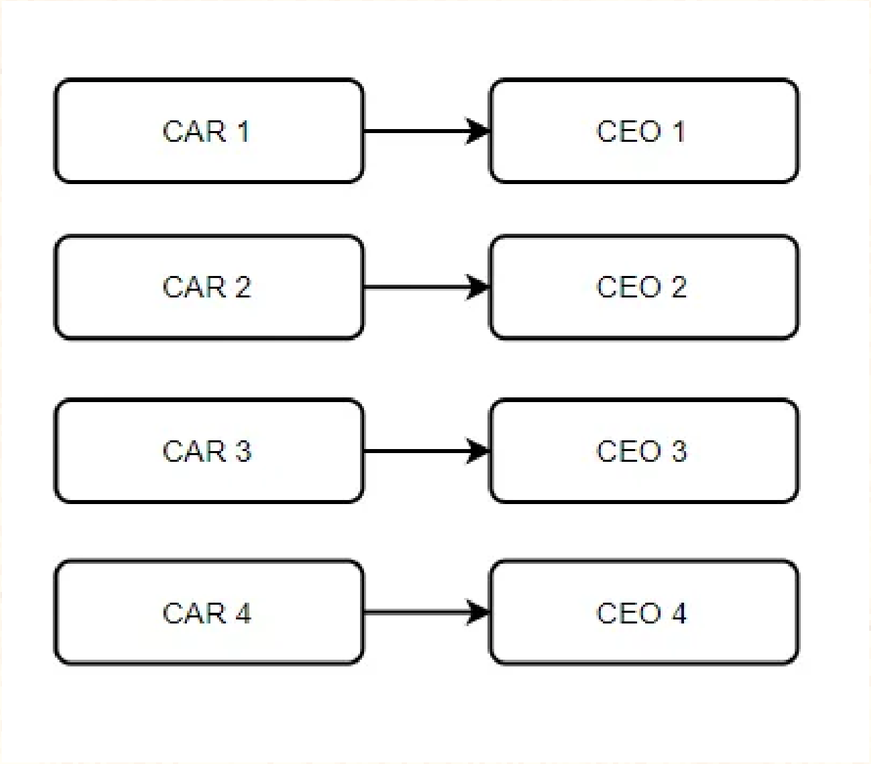
ONETOONEFIELD





ONETOONEFIELD

- Car company can have only one CEO.
- CEO can work only one Car company.



ONETOONEFIELD

```
Syntax:- OneToOneField(to, on_delete, parent_link=False, **options)
```

1. **to (required): It should be a reference to the related model class.**
2. **on_delete (required): This parameter specifies what should happen when the referenced object is deleted.**
    - **models.CASCADE: Deletes the object that has the OneToOneField when the referenced object is deleted.**
    - **models.PROTECT: Prevents deletion of the referenced object by raising a ProtectedError.**
    - **models.SET_NULL: Sets the OneToOneField to NULL when the referenced object is deleted.**
    - **models.SET_DEFAULT: Sets the OneToOneField to its default value when the referenced object is deleted.**
    - **models.SET(): Sets the OneToOneField to the value passed as an argument when the referenced object is deleted.**
    - **models.SET_ON_DELETE: Sets the OneToOneField to the value of the field's on_delete attribute when the referenced object is deleted.**
    - **models.DO_NOTHING: Takes no action when the referenced object is deleted. It assumes that you handle the situation manually (rarely used).**

ONETOONEFIELD

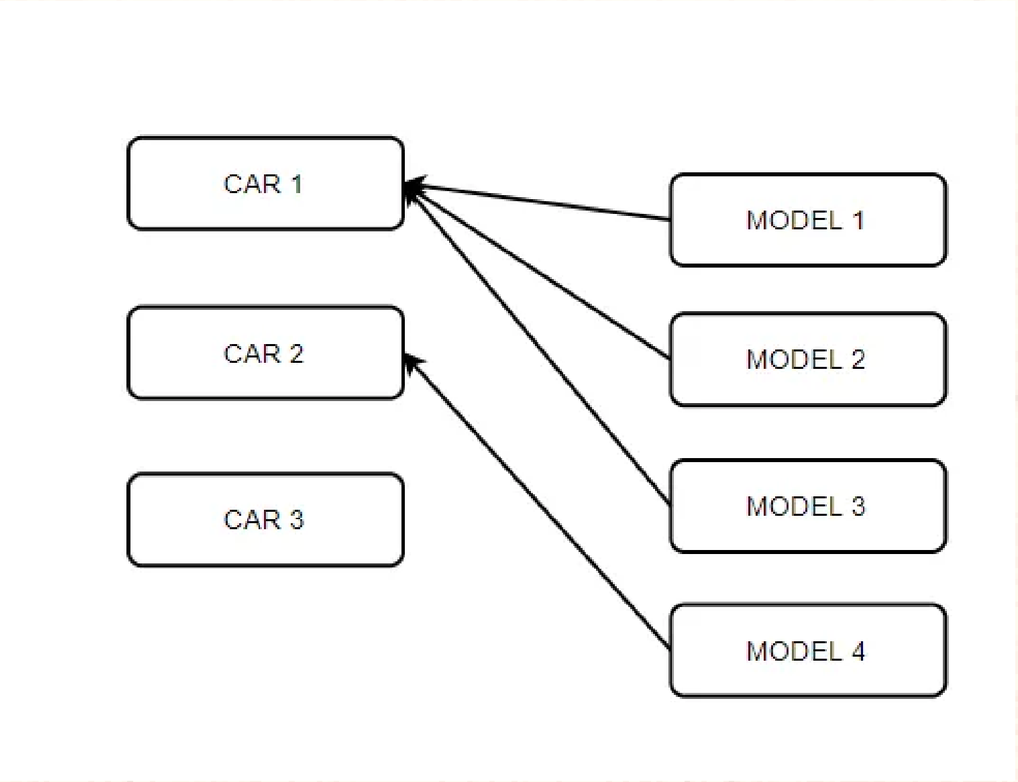
```python
from django.db import models

class Car(models.Model):
    name = models.CharField(max_length=255)

class Ceo(models.Model):
    name = models.CharField(max_length=255, blank=True)
    car = models.OneToOneField(Car, on_delete=models.CASCADE)
```

ONETOONEFIELD

1. **related_name:** This parameter allows you to specify the reverse relation from the related model back to the model with the OneToOneField. It defines the name to use for the reverse relation. If not provided, Django automatically generates a default name based on the name of the model.
2. **related_query_name:** This parameter specifies the name to use for the reverse relation's query name. It is used in the case of a related model being referenced in a query.
3. **db_constraint:** When set to True, this parameter enforces a database-level constraint to ensure the uniqueness of the relationship. It creates a foreign key constraint in the database.
4. **unique:** When set to True, this parameter ensures that the relationship is unique. Only one instance of the related model can be associated with a specific instance of the model containing the OneToOneField.
5. **null:** When set to True, this parameter allows the OneToOneField to be set to NULL if the related object is not provided. By default, it is set to False, and a related object must be specified.

ONE TO MANY

1. **- Car company can have one or more car model.**
2. **- Car model can only belong to one car company.**

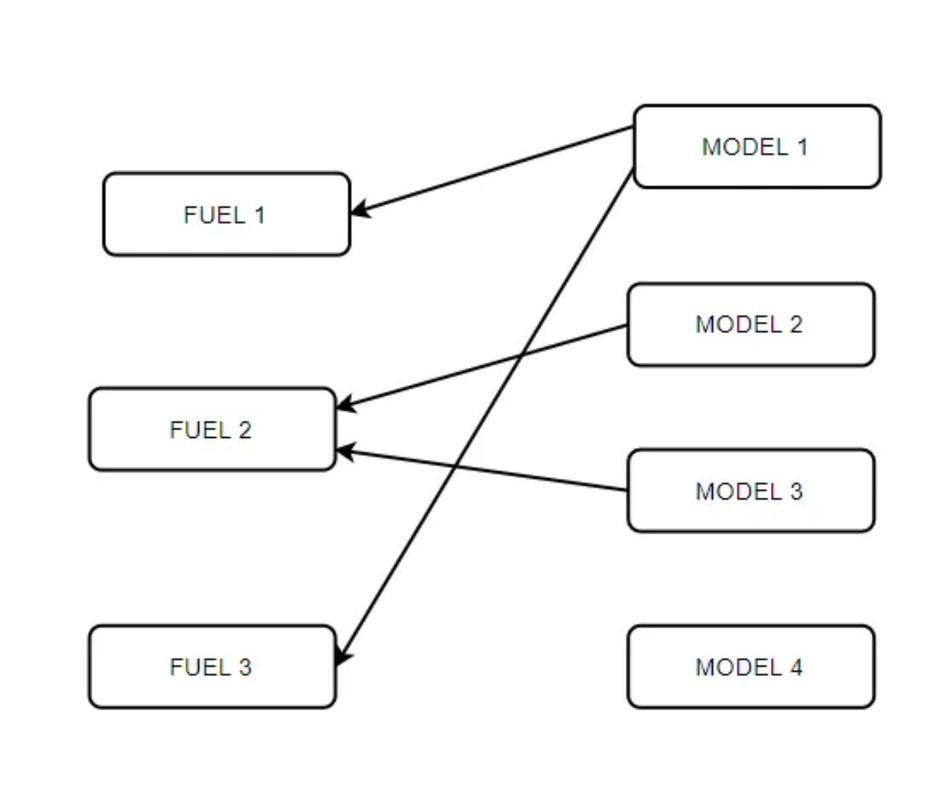


ONE TO MANY

```python
from django.db import models

class Car(models.Model):
    name = models.CharField(max_length=255)

class CarModel(models.Model):
    name = models.CharField(max_length=255)
    car = models.ForeignKey(Car, on_delete=models.CASCADE)
```

MANY TO MANY

- Many car models can run on same fuel type.
- Car model can run on diffrent fuel types.



FOR OUR PROJECT

```
Author (One-to-Many with Posts): An author can have multiple blog posts.
Category (Many-to-Many with Posts): A blog post can belong to multiple categories.
Profile (One-to-One with User): Each user has a single profile.
```


THANKS