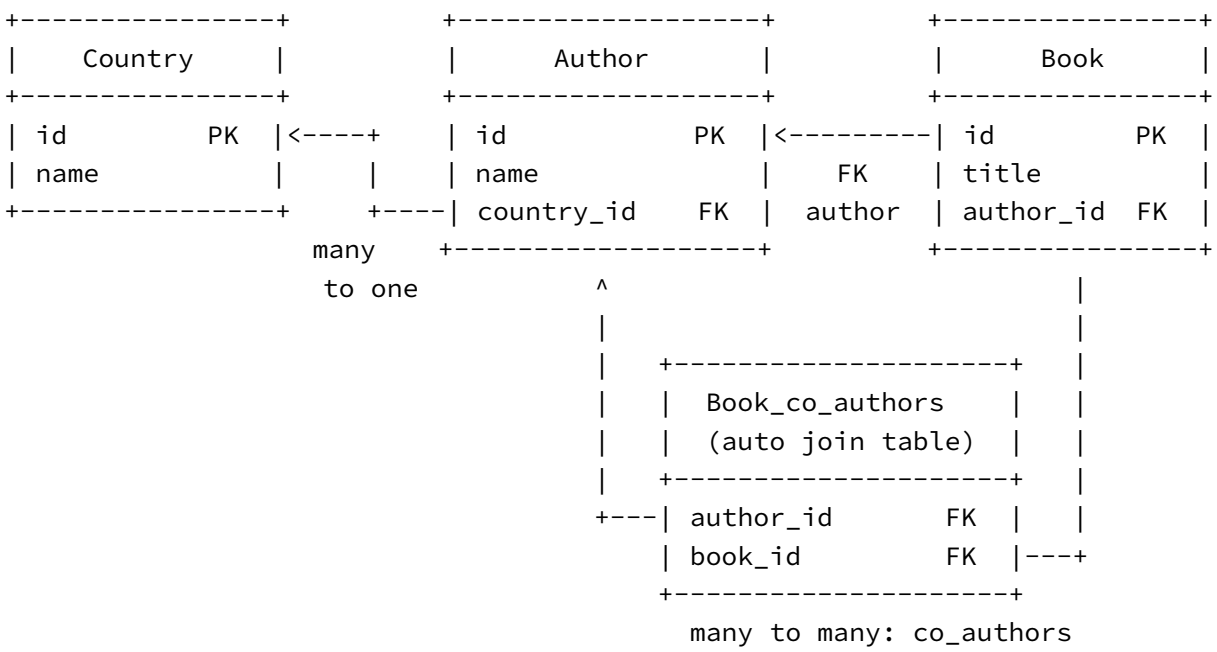
N+1 queries in Django Admin

The situation

Let's say your Django admin application comes with 3 models - Author, Book and Country. Those models are obviously related where a book can have one main author, multiple co-authors and an author can be only from one country.

The table scheme goes like:



Now you want to createa Django Admin for the Book model.

The problem

Your `Author` model string-represents itself accorging to following method:

```
class Author(models.Model):
    # ...

    def __str__(self):
        return f"{self.name} ({self.country})"
```

Soon you realize that your Book model admin page generates a lot of SQL N+1 queries. Book models refers to `Author` model which is fully fetched. But the `Author` model defines `__str__` method, which works with `country` (column `country_id`) property which is another FK to `Country`

model, which wasn't fetched. Why? When you edit a model in Django Admin which relations has further relations to other models you want to work with (like use their properties) you'd typically use `select_related()` or `prefetch_related()` methods. Without them properties of related models are not fetched. But the thing is you need to control querysets behind dropdowns, not the edited object queryset.

The solution

Django admin comes with a solution which is `formfield_for_foreignkey()` and `formfield_for_manytomany()` methods. These methods (as its name suggests) are made as hooks for form fields. Such form fields are mirror of `ForeignKey` or `ManyToManyField` model fields. You can step in, modify the query for `author` and `co_authors` fields with known `*_related()` methods and watch, how number of SQL queries drops.

```
from django.contrib import admin
from .models import Author, Book

@admin.register(Book)
class BookAdmin(admin.ModelAdmin):

    def formfield_for_foreignkey(self, db_field, request, **kwargs):
        if db_field.name == "author":
            qs = self.get_field_queryset(kwargs.get("using"), db_field,
↪   request)

            if qs is None:
                qs = Author.objects.all()

            kwargs["queryset"] = qs.select_related("country")

        return super().formfield_for_foreignkey(db_field, request, **kwargs)

    def formfield_for_manytomany(self, db_field, request, **kwargs):
        if db_field.name == "co_authors":
            qs = self.get_field_queryset(kwargs.get("using"), db_field,
↪   request)

            if qs is None:
                qs = Author.objects.all()

            kwargs["queryset"] = qs.select_related("country")
```

```
        return super().formfield_for_manytomany(db_field, request, **kwargs)
```