Django model constraints

Django models and forms provide various options how to validate incoming data and catch data inconsistency. One can set up such things like:

- form field validations
- form complex (multi-field) validations
- model validations
- model complex (multi-field) validations
- additional manual validations in view

Those are okay, but not perfect for all possible cases. The reason is form validators can be bypassed in cases where forms and ORM models are not used - like admin (assume here you did not specify extra form for admin) or - and the most likely - direct manipulation with the database. Like dump restoration or import from another (3rd party) tool - like psql. Those cases are out of Django app control and the only way is to have some kind of "validations" directly in the database. That's when constraints come to the game.

Constraints

Django does support model constraints. That means that any constraint defined on a model is later reflected to the database via database migrations and sits there. If you attempt to violate your rules by inserting directly into the database you will get caught.

Let's assume this simple model which represents Facebook's OG tags.

```
class OgTag(models.Model):
    page = PageField(blank=True, null=True )
    url = models.CharField(blank=True, max_length=255)
    name = models.CharField(max_length=50)
    value = models.TextField(blank=True, null=True)
    image = FilerImageField(blank=True, null=True, on_delete=models.CASCADE)
    is_default = models.BooleanField(default=False)

    objects = OgTagManager()

    def get_image(self):
        if self.image:
            return self.image.url
```

Let's assume the following rules:

- `page` and `url` are similar fields
- `page` is a list of site pages and `url` is for arbitrary URL
- both are mutually exclusive
- but one of them is required
- `is_default` is unique in a combination with `value`
- `value` and `image` are mutually exclusive
- but one of them is required

Those rules can be enforced quite easily with form `clean()` validation method. But you can also set up model constraints that will be in sync with your database:

```
class OgTag(models.Model):
    ...

    class Meta:
        constraints = [
            models.CheckConstraint(check=~Q(value="") |
↪   Q(image__isnull=False), name="image_or_value"),
            models.CheckConstraint(check=Q(value="") |
↪   Q(image__isnull=True), name="image_or_value_not_both"),
            models.CheckConstraint(check=~Q(url="") | Q(page__isnull=False),
↪   name="page_or_url"),
            models.CheckConstraint(check=Q(url="") | Q(page__isnull=True),
↪   name="page_or_url_not_both"),
            models.UniqueConstraint(fields=["name"],
↪   condition=Q(is_default=True), name="unique_defaults"),
        ]
```

Now you just need to run database migrations and you are all set up.

Error messages

One (big) downside that comes with constraints is that Django cannot catch, parse and map database errors to some sane error messages, so you just get regular database errors which lead to HTTP 5XX error code. To prevent this, you need to reflect those constraint rules to a form validation where you can construct user-friendly error messages. Then you are fully set up.