

Writing Safe Database Migrations in Django

A photograph of a natural landscape in autumn. In the foreground, a stack of several flat, grey stones is balanced on top of a weathered wooden stump. The background is a dense field of low-lying plants and shrubs with leaves in various shades of green, yellow, and orange, suggesting a transition between seasons. The overall lighting is soft and natural.



Hi, I'm

Markus Holtermann



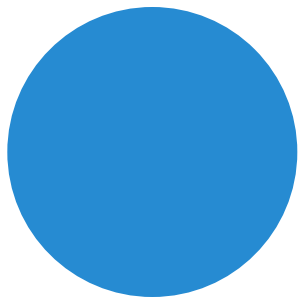
django

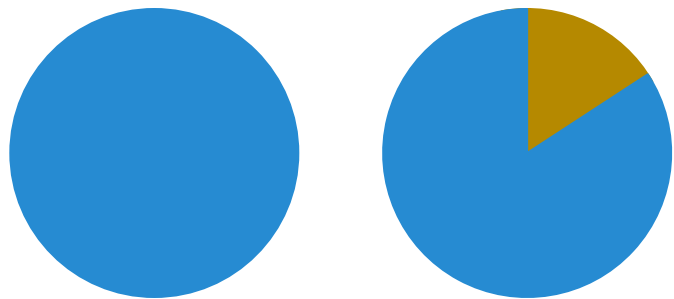
Don't Be Afraid Of Writing Migrations

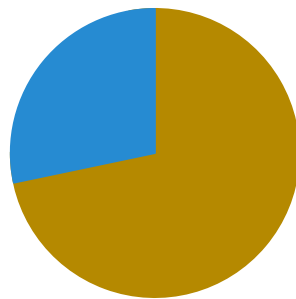
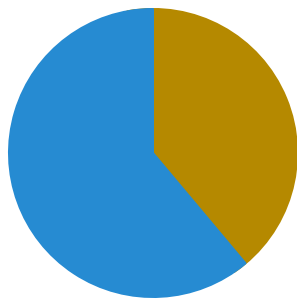
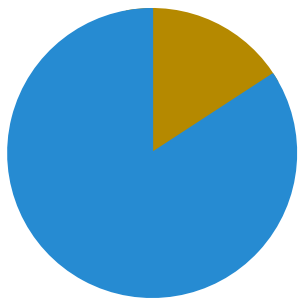
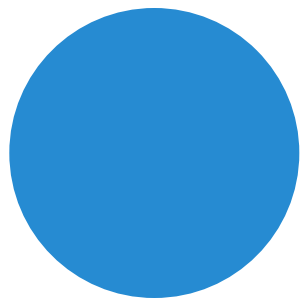
<https://markusholtermann.eu/2016/04/dont-be-afraid-of-writing-migrations/>

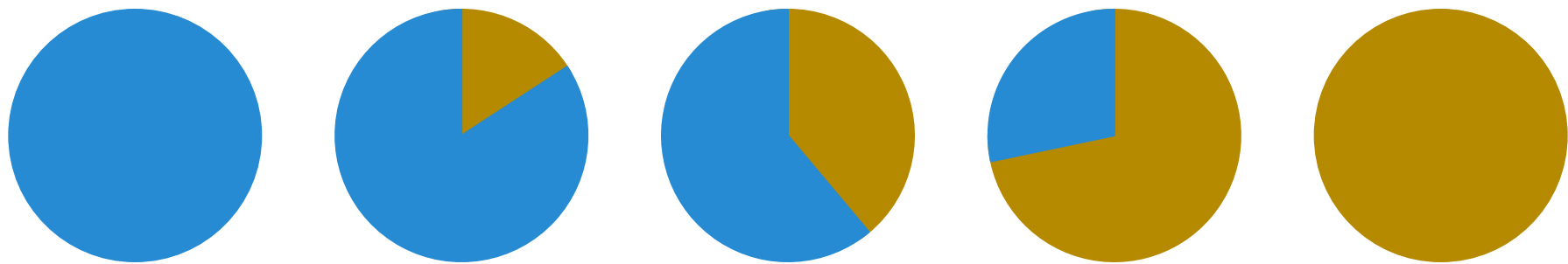
The background of the slide is a dark, textured surface covered with numerous small, glowing lanterns. These lanterns have a warm, yellow-orange light emanating from within, and their frames appear to be made of a dark, possibly metal, material with intricate patterns. The lanterns are arranged in a way that creates a sense of depth and repetition, filling the entire frame with a soft, ambient glow.

Migrations! How? When?











When Do We Deploy Migrations?



How Do We Deploy Migrations?

A photograph of a forest path covered in brown leaves, lined with trees that have bright green foliage. In the distance, a wooden tower or structure is visible through the trees.

**Only Go Forwards &
Never Look Back**

DjangoCon Europe 2021 • @m_holtermann

```
#!/bin/sh
```

```
set -e
```

```
cmd="$@"
```

```
until django-admin dbshell -- --command '\q'; do  
    >&2 echo "Postgres is unavailable - sleeping "  
    sleep 1  
done
```

```
>&2 echo "Postgres is up - executing command "
```

```
django-admin migrate -v 2
```

```
exec $cmd
```



Adding A Field Is Harmless

```
from django.db import models
```

```
class AddFieldModel1(models.Model):  
    name = models.CharField(  
        max_length=10)
```

```
class AddFieldModel2(models.Model):  
    name = models.CharField(  
        max_length=10)
```

```
from django.db import models
```

```
class AddFieldModel1(models.Model):  
    name = models.CharField(  
        max_length=10)  
    field = models.CharField(  
        max_length=10, null=True)
```

```
class AddFieldModel2(models.Model):  
    name = models.CharField(  
        max_length=10)  
    field = models.CharField(  
        default="aaaaaaaaaa",  
        max_length=10)
```

```
from django.db import migrations, models

class Migration(migrations.Migration):

    dependencies = [
        ("add_field", "0001_initial"),
    ]

    operations = [
        migrations.AddField(
            model_name="addfieldmodel1",
            name="field",
            field=models.CharField(max_length=10, null=True),
        ),
        migrations.AddField(
            model_name="addfieldmodel2",
            name="field",
            field=models.CharField(default="aaaaaaaaaa", max_length=10),
        ),
    ]
```

```
BEGIN;

--
-- Add field field to addfieldmodel1
--
ALTER TABLE "add field addfieldmodel1 "
    ADD COLUMN "field" varchar(10) NULL;

--
-- Add field field to addfieldmodel2
--
ALTER TABLE "add field addfieldmodel2 "
    ADD COLUMN "field" varchar(10) DEFAULT 'aaaaaaaaaa' NOT NULL;
ALTER TABLE "add field addfieldmodel2 "
    ALTER COLUMN "field" DROP DEFAULT;

COMMIT;
```



But I Want A Default Value!

```
from django.core.management.base import BaseCommand
from django.db import transaction

from safe_migrations.add_field.models import AddFieldModel1

CHUNK_SIZE = 5000

class Command(BaseCommand):
    def handle(self, *args, **options):
        updated = 0
        while updated < CHUNK_SIZE:
            with transaction.atomic():
                ids = (
                    AddFieldModel1.objects.filter(field__isnull=True)
                    .select_for_update()
                    .values_list("id", flat=True)[:CHUNK_SIZE]
                )
                updated = AddFieldModel1.objects.filter(id__in=ids).update(
                    field="bbbbbbbbbb"
                )
```

Adding An Index

```
from django.db import models
```

```
class AddIndexModel1(models.Model):  
    name = models.CharField(  
        max_length=10)
```

```
class AddIndexModel2(models.Model):  
    name = models.CharField(  
        max_length=10)
```

```
from django.db import models
```

```
class AddIndexModel1(models.Model):  
    name = models.CharField(  
        max_length=10, db_index=True)
```

```
class AddIndexModel2(models.Model):  
    name = models.CharField(  
        max_length=10)
```

```
class Meta:  
    indexes = [  
        models.Index(  
            fields=("name",),  
            name="my_idx")  
    ]
```

```
from django.db import migrations, models

class Migration(migrations.Migration):

    dependencies = [
        ("add_index", "0001_initial"),
    ]

    operations = [
        migrations.AlterField(
            model_name="addindexmodel1",
            name="name",
            field=models.CharField(db_index=True, max_length=10),
        ),
        migrations.AddIndex(
            model_name="addindexmodel2",
            index=models.Index(fields=["name"], name="my_idx"),
        ),
    ]
```

```
BEGIN;

--
-- Alter field name on addindexmodell1
--
CREATE INDEX "add index addindexmodell1 name_adf72323"
  ON "add index addindexmodell1" ("name");
CREATE INDEX "add index addindexmodell1 name adf72323 like"
  ON "add_index_addindexmodell1" ("name" varchar_pattern_ops);

--
-- Create index my_idx on field(s) name of model addindexmodell2
--
CREATE INDEX "my_idx" ON "add_index_addindexmodell2" ("name");

COMMIT;
```

```
from django.contrib.postgres.operations import AddIndexConcurrently
from django.db import migrations, models
```

```
class Migration(migrations.Migration):
```

```
    atomic = False
```

```
    dependencies = [
        ("add_index", "0001_initial"),
    ]
```

```
    operations = [
        migrations.AlterField(
            model name="addindexmodel1",
            name="name",
            field=models.CharField(db_index=True, max_length=10),
        ),
        AddIndexConcurrently (
            model name="addindexmodel2",
            index=models.Index(fields=["name"], name="my_idx"),
        ),
    ]
```

~~BEGIN;~~

--

-- Alter field name on addindexmodell1

--

CREATE INDEX "add index addindexmodell1 name_adf72323"

ON "add index addindexmodell1" ("name");

CREATE INDEX "add index addindexmodell1 name adf72323 like"

ON "add_index_addindexmodell1" ("name" varchar_pattern_ops);

--

-- Create index my_idx on field(s) name of model addindexmodel2

--

CREATE INDEX CONCURRENTLY "my_idx" ON "add_index_addindexmodel2" ("name");

~~COMMIT;~~



Test Your Migrations

DjangoCon Europe 2021 • @m_holtermann

Summary

- Apply migrations before you deploy
- Only go forwards & never look back
- Only add nullable fields
- Populate default value with management command
- Add indexes concurrently
- Use meaningful index names
- Have working backups of your database
- Test (complex) on production-like data



Thanks!

DjangoCon Europe 2021 • @m_holtermann

Sources

- <https://www.postgresql.org/docs/10/sql-altertable.html>
- <https://www.postgresql.org/docs/11/sql-altertable.html>
- <https://www.citusdata.com/blog/2018/02/22/seven-tips-for-dealing-with-postgres-locks/>
- <https://docs.djangoproject.com/en/3.2/ref/contrib/postgres/operations/#concurrent-index-operations>