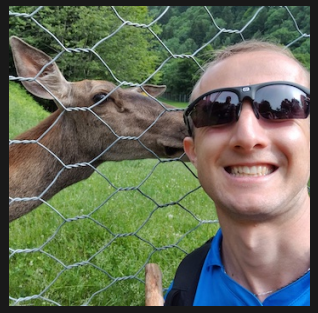
Django migrations friend or foe?

Optimize them for testing

FOSDEM 2024

WHO AM I



Denny Biasiolli

Full Stack Developer
(JavaScript, Python, Go)

Front End Developer UX/ UI
Fingerprint Compliance Services Ltd.

Italy, Savigliano (CN)

denny.biasiolli@gmail.com

www.dennybiasiolli.com

@dennybiasiolli

MIGRATIONS

Way to propagate changes to models into a database schema.

https://docs.djangoproject.com/en/4.2/topics/migrations/

MIGRATION COMMANDS

- 👨‍🔧 makemigrations
- 👨‍💻 migrate
- 🕵️‍♂️ showmigrations
- 👨‍🔬 sqlmigrate

https://docs.djangoproject.com/en/4.2/topics/migrations/#the-commands

`makemigrations`

Creates new migration(s) for apps.

```
manage.py makemigrations [--empty] [-n NAME] [app_label]
```

- `--empty`
  Create an empty migration.
- `-n NAME, --name NAME`
  Use this name for migration file(s)
- `app_label`

https://docs.djangoproject.com/en/4.2/ref/django-admin/#django-admin-makemigrations

makemigrations

Add/edit models

```
# main/models.py
from django.contrib.auth import get_user_model
from django.db import models


class Tweet(models.Model):
    created_by = models.ForeignKey(
      get_user_model(), on_delete=models.CASCADE)
    created_at = models.DateTimeField(auto_now_add=True)
    text = models.CharField(max_length=140)
```

makemigrations

Create migrations

```
$ manage.py makemigrations

Migrations for 'main':
  main/migrations/0001_initial.py
    - Create model Tweet
```

makemigrations

Inspect migration files

```
# Generated by Django 4.1.4 on 2023-01-24 16:00

from django.db import migrations, models
# ...
```

```
class Migration(migrations.Migration):

    initial = True

    dependencies = [
        migrations.swappable_dependency(settings.AUTH_USER_MOD
    ]

    operations = [
        migrations.CreateModel(
            # ...
        ),
    ]
```

`migrate`

Updates database schema.

```
manage.py migrate [app_label] [migration_name]
```

- `app_label`
- `migration_name`
  Database state will be brought to the state after that migration. Use the name "zero" to unapply all migrations.

https://docs.djangoproject.com/en/4.2/ref/django-admin/#django-admin-migrate

`migrate`

First migration

```
$ manage.py migrate

Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  ...
  Applying auth.0012_alter_user_first_name_max_length... OK
  Applying sessions.0001_initial... OK
```

`migrate`

Rollback migrations

```
$ manage.py migrate admin zero

Operations to perform:
  Unapply all migrations: admin
Running migrations:
  Rendering model states... DONE
  Unapplying admin.0003_logentry_add_action_flag_choices... OK
  Unapplying admin.0002_logentry_remove_auto_add... OK
  Unapplying admin.0001_initial... OK
```

`migrate`

Move to a specific migration

```
# manage.py migrate admin 0002_logentry_remove_auto_add
$ manage.py migrate admin 0002

Operations to perform:
  Target specific migration: 0002_logentry_remove_auto_add,
    from admin
Running migrations:
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
```

`migrate`

Applying missing migrations

```
$ manage.py migrate admin

Operations to perform:
  Apply all migrations: admin
Running migrations:
  Applying admin.0003_logentry_add_action_flag_choices... OK
```

```
$ manage.py migrate

Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions
Running migrations:
  No migrations to apply.
```

MIGRATIONS, UNDER THE HOOD

```
mydatabase# \d django_migrations
 Table "public.django_migrations"
 Column  |           Type
---------+--------------------------
 id      | bigint
 app     | character varying(255)
 name    | character varying(255)
 applied | timestamp with time zone
```

MIGRATIONS, UNDER THE HOOD

```
mydatabase=# select * from django_migrations;
 id |     app      |                          name
----+--------------+-----------------------------------------
  1 | contenttypes | 0001_initial
  2 | auth         | 0001_initial
  3 | admin        | 0001_initial
  4 | admin        | 0002_logentry_remove_auto_add
  5 | admin        | 0003_logentry_add_action_flag_choices
  6 | contenttypes | 0002_remove_content_type_name
  7 | auth         | 0002_alter_permission_name_max_length
  8 | auth         | 0003_alter_user_email_max_length
  9 | auth         | 0004_alter_user_username_opts
 10 | auth         | 0005_alter_user_last_login_null
 11 | auth         | 0006_require_contenttypes_0002
 12 | auth         | 0007_alter_validators_add_error_messages
```

`showmigrations`

Shows all available migrations for the current project

```
manage.py showmigrations [app_label]
```

https://docs.djangoproject.com/en/4.2/ref/django-admin/#django-admin-showmigrations

showmigrations

```
$ manage.py showmigrations main

main
 [X] 0001_initial
```

sqlmigrate

Prints the SQL statements for the named migration.

```
manage.py sqlmigrate app_label migration_name
```

https://docs.djangoproject.com/en/4.2/ref/django-admin/#django-admin-sqlmigrate

sqlmigrate

```
$ manage.py sqlmigrate main 0001
```

```
BEGIN;
--
-- Create model Tweet
--
CREATE TABLE "main_tweet" ("id" bigint NOT NULL PRIMARY KEY GE
ALTER TABLE "main_tweet" ADD CONSTRAINT "main_tweet_created_by
CREATE INDEX "main_tweet_created_by_id_de58f942" ON "main_twee
COMMIT;
```

CHANGING MODELS

Edit model

```
+++ main/models.py
 class Tweet(models.Model):
     created_by = models.ForeignKey(get_user_model(), on_delet
     created_at = models.DateTimeField(auto_now_add=True)
-    text = models.CharField(max_length=140)
+    text = models.CharField(max_length=250)
```

Create migration

```
$ manage.py makemigrations main

Migrations for 'main':
  main/migrations/0002_alter_tweet_text.py
    - Alter field text on tweet
```

CHANGING MODELS

Inspect migration

```
# main/migrations/0002_alter_tweet_text.py
class Migration(migrations.Migration):

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

    operations = [
        migrations.AlterField(
            model_name="tweet",
            name="text",
            field=models.CharField(max_length=250),
        ),
    ]
```

CHANGING MODELS

Show SQL statement

```
$ manage.py sqlmigrate main 0002
```

```sql
BEGIN;
--
-- Alter field text on tweet
--
ALTER TABLE "main_tweet" ALTER COLUMN "text" TYPE varchar(250)
COMMIT;
```

CHANGING MODELS

Apply migration

```
$ manage.py migrate main

Operations to perform:
  Apply all migrations: main
Running migrations:
  Applying main.0002_alter_tweet_text... OK
```

FURTHER CHANGES?

FURTHER CHANGES?

- enabling tweet likes
  (adding `Like` model)

FURTHER CHANGES?

- enabling tweet likes
  (adding `Like` model)
- enabling retweets
  (nullable `text` field and `related_tweet` field)

FURTHER CHANGES?

- enabling tweet likes
  (adding `Like` model)
- enabling retweets
  (nullable `text` field and `related_tweet` field)
- forgot the `related_name` for `Like.tweet` field

FURTHER CHANGES?

- enabling tweet likes
  (adding `Like` model)
- enabling retweets
  (nullable `text` field and `related_tweet` field)
- forgot the `related_name` for `Like.tweet` field
- enabling followers
  (`Follow` model)

SHOW MIGRATIONS

```
$ manage.py showmigrations main

main
 [X] 0001_initial
 [X] 0002_alter_tweet_text
 [ ] 0003_like
 [ ] 0004_tweet_related_tweet_alter_tweet_text
 [ ] 0005_alter_like_tweet
 [ ] 0006_follow
```

ADD `shop` APP

- `Customer` model and shipping details

ADD `shop` APP

- `Customer` model and shipping details
- adding `is_premium` field to `Customer` model

ADD `shop` APP

- `Customer` model and shipping details
- adding `is_premium` field to `Customer` model
- creating dedicated `ShippingAddress` model

ADD `shop` APP

- `Customer` model and shipping details
- adding `is_premium` field to `Customer` model
- creating dedicated `ShippingAddress` model
- migrating data to new shipping addresses

ADD `shop` APP

- `Customer` model and shipping details
- adding `is_premium` field to `Customer` model
- creating dedicated `ShippingAddress` model
- <u>migrating data to new shipping addresses</u>
- removing Customer shipping fields
  (one migration per field: state, province, city, zip code, address, name)

FURTHER CHANGES?

FURTHER CHANGES?

- increasing length of `ShippingAddress` fields

FURTHER CHANGES?

- increasing length of `ShippingAddress` fields
- adding `Order` model

FURTHER CHANGES?

- increasing length of `ShippingAddress` fields
- adding `Order` model
- adding `created_at` field to `Order` model

FURTHER CHANGES?

- increasing length of `ShippingAddress` fields
- adding `Order` model
- adding `created_at` field to `Order` model
- adding `OrderLine` model and manager

FURTHER CHANGES?

- increasing length of `ShippingAddress` fields
- adding `Order` model
- adding `created_at` field to `Order` model
- adding `OrderLine` model and manager
- adding `customer_type` choice field
  ("Free" and "Premium")

FURTHER CHANGES?

- increasing length of `ShippingAddress` fields
- adding `Order` model
- adding `created_at` field to `Order` model
- adding `OrderLine` model and manager
- adding `customer_type` choice field
  ("Free" and "Premium")
- adding `customer_type` migration from `is_premium`

FURTHER CHANGES?

- increasing length of `ShippingAddress` fields
- adding `Order` model
- adding `created_at` field to `Order` model
- adding `OrderLine` model and manager
- adding `customer_type` choice field
  ("Free" and "Premium")
- adding `customer_type` migration from `is_premium`
- removing is_premium field

FURTHER CHANGES?

- increasing length of `ShippingAddress` fields
- adding `Order` model
- adding `created_at` field to `Order` model
- adding `OrderLine` model and manager
- adding `customer_type` choice field
  ("Free" and "Premium")
- adding `customer_type` migration from `is_premium`
- removing is_premium field
- adding more customer types
  ("Bronze", "Silver", "Gold", "Platinum")

FURTHER CHANGES?

- increasing length of `ShippingAddress` fields
- adding `Order` model
- adding `created_at` field to `Order` model
- adding `OrderLine` model and manager
- adding `customer_type` choice field
  ("Free" and "Premium")
- adding `customer_type` migration from `is_premium`
- removing is_premium field
- adding more customer types
  ("Bronze", "Silver", "Gold", "Platinum")
- renaming `product_quantity` to `quantity`

FURTHER CHANGES?

- increasing length of `ShippingAddress` fields
- adding `Order` model
- adding `created_at` field to `Order` model
- adding `OrderLine` model and manager
- adding `customer_type` choice field
  ("Free" and "Premium")
- adding `customer_type` migration from `is_premium`
- removing is_premium field
- adding more customer types
  ("Bronze", "Silver", "Gold", "Platinum")
- renaming `product_quantity` to `quantity`
- adding Product model

MIGRATIONS?

```
$ manage.py showmigrations shop
 [ ] 0001_initial
 [ ] 0002_customer_is_premium
 [ ] 0003_shippingaddress
 [ ] 0004_migrate_shipping_address
 [ ] 0005_remove_customer_shipping_state
 [ ] 0006_remove_customer_shipping_province
 [ ] 0007_remove_customer_shipping_city
 [ ] 0008_remove_customer_shipping_zip_code
 [ ] 0009_remove_customer_shipping_address
 [ ] 0010_remove_customer_shipping_name
 [ ] 0011_alter_shippingaddress_address_and_more
 [ ] 0012_order
 [ ] 0013_order_created_at
 [ ] 0014_orderline
 [ ] 0015_alter_customer_user
 [ ] 0016_customer_customer_type
 [ ] 0017_migrate_is_premium_to_customer_type
 [ ] 0018_remove_customer_is_premium
 [ ] 0019_alter_customer_customer_type
 [ ] 0020_alter_customer_customer_type
 [ ] 0021_alter_customer_customer_type
 [ ] 0022_alter_customer_customer_type
 [ ] 0023_alter_customer_customer_type
 [ ] 0024_alter_customer_customer_type
 [ ] 0025_rename_product_quantity_orderline_quantity
 [ ] 0026_product
```

WHAT ABOUT PERFORMANCES?

DISCLAIMER

Timing may change from laptop to laptop.

⏱ TEST PERFORMANCES

20x apps like shop

```
$ manage.py test

Found 152 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
.................................................
----------------------------------------------------
Ran 152 tests in 0.924s 😀

OK
Destroying test database for alias 'default'...
```

⏱ TEST PERFORMANCES

```
$ time manage.py test

Ran 152 tests in 0.845s
19.91s user 0.31s system 99% cpu 20.409 total
```

Command	Execution time
Creating test database	~20s 😕
Running tests	~1s

A POSSIBLE WORKAROUND

```
$ manage.py test --keepdb
```

- preserve the test database between runs
- if (the database) does not exist, it will first be created
- migrations will also be applied in order to keep it up to date

`--keepdb` pros and cons

- 🎉 Saves ~20s for each test run after the first one
- 😥 Not easy to configure in CI/CD
  - cache/artifacts in GitHub workflows
  - external test DB

ANOTHER WORKAROUND

Django settings

```
MIGRATE = False  # default to True
```

When set to False, migrations won’t run when creating the test database. This is similar to setting None as a value in MIGRATION_MODULES, but for all apps.

https://docs.djangoproject.com/en/4.2/ref/settings/#migrate

`MIGRATE = False` pros and cons

- 🎉 Single line change in your codebase
- 🎉 Doesn't run migrations during tests
- 😥 It's like `makemigrations` + `migrate` before running tests
- 😥 ~+5s in our test repository

⏱ PERFORMANCES

	Creating test DB	Running tests
Before	~20s	~1s
--keepdb	~0s 🥳	~1s
MIGRATE = False	~25s 😑	~1s

squashmigrations

```
manage.py squashmigrations \
    [--no-optimize]  # disable merging of CreateModel and AddF
    app_label [start_migration_name] migration_name
```

Squash an existing set of migrations
into a single new one.

- `--no-optimize`
  disable the optimizer when generating a squashed migration.
  Example: disable merge of `AddField` commands
  placed right after a `CreateModel` for the same table

https://docs.djangoproject.com/en/4.2/ref/django-admin/#django-admin-squashmigrations

squashmigrations

Applied to shop app

```
$ python manage.py squashmigrations shop 0026

Will squash the following migrations:
 - 0001_initial
 - 0002_customer_is_premium
 - ...
 - 0026_product
Do you wish to proceed? [yN] y
```

squashmigrations

```
Optimizing...
  Optimized from 29 operations to 27 operations.
```

```
Created new squashed migration 0001_squashed_0026_product.py
  You should commit this migration but leave the old ones in
  place; the new migration will be used for new installs.
  Once you are sure all instances of the codebase have
  applied the migrations you squashed, you can delete them.
```

```
Manual porting required
  Your migrations contained functions that must be manually
  copied over, as we could not safely copy their
  implementation.
  See the comment at the top of the squashed migration
  for details.
```

squashmigrations

Inspecting migration file

```
# 0001_squashed_0026_product.py

# Functions from the following migrations need manual copying.
# Move them and any dependencies into this file, then update
# the RunPython operations to refer to the local versions:
# shop.migrations.0004_migrate_shipping_address
# shop.migrations.0017_migrate_is_premium_to_customer_type
```

```
  migrations.RunPython(
-     code=shop.migrations.0004_migrate_shipping_address.forwa
+     code=0004_forward_func,
-     reverse_code=shop.migrations.0004_migrate_shipping_addre
+     reverse_code=0004_backward_func,
  ),
```

squashmigrations

Inspecting migration file

```
class Migration(migrations.Migration):

    replaces = [
        ("shop", "0001_initial"),
        ("shop", "0002_customer_is_premium"),
        # ...
        ("shop", "0026_product"),
    ]
```

RECOMMENDED PROCESS

1. squash, keeping the old files, commit and release
2. wait until all systems are upgraded with the new release
3. remove the old migration files, commit and do a second release

RECOMMENDED PROCESS

4. transition the squashed migration to a normal migration:

- delete all the migration files it replaces
- update all migrations that depend on the deleted migrations to depend on the squashed migration instead
- remove the `replaces` attribute in the squashed migration

New in Django 4.1.

PRUNING REFERENCES TO DELETED MIGRATIONS

If it is likely that you may reuse the name of a deleted migration in the future, you should remove references to it from Django's migrations table with

```
manage.py migrate --prune
```

TEST PERFORMANCES AFTER SQUASHING

```
$ time manage.py test

Ran 152 tests in 0.948s
20.47s user 0.35s system 99% cpu 21.001 total
```

	Creating test DB	Running tests
Before	~20s	~1s
--keepdb	~0s 🥳	~1s
MIGRATE = False	~25s 😑	~1s
After squashing	~20s 😑	~1s


imgflip.com

WHAT'S THE POINT?

Move back
from having several hundred migrations
to just a few


I WANTED TO
SPEED UP TESTS!
imgflip.com

Do you really want to speed up database creation in tests?

Re-creating migrations from scratch and doing a lot of manual tasks?

RECREATE MIGRATIONS

1. annotate migrations for a specific app

```
$ APP_LABEL=shop
$ manage.py showmigrations $APP_LABEL
shop
 [X] 0001_initial
 [X] 0002_customer_is_premium
 [X] 0003_shippingaddress
 # ...
 [X] 0026_product
```

RECREATE MIGRATIONS

2. create a python list with this format

```
# "shop" is the app_label
replaces = [
    ("shop", "0001_initial"),
    ("shop", "0002_customer_is_premium"),
    ("shop", "0003_shippingaddress"),
    # ...
    ("shop", "0026_product"),
]
```

RECREATE MIGRATIONS

3. move migrations in a temporary directory

```
$ mv $APP_LABEL/migrations $APP_LABEL/old_migrations
```

make sure that migrations are no longer there

```
$ manage.py showmigrations $APP_LABEL
shop
 (no migrations)
```

RECREATE MIGRATIONS

4. recreate first migration from scratch

using a different name than the old migration 0001

```
$ manage.py makemigrations $APP_LABEL --name=init_squashed
Migrations for 'shop':
  shop/migrations/0001_init_squashed.py
    - Create model Customer
    - Create model Order
    - Create model ShippingAddress
    - Create model Product
    - Create model OrderLine
```

RECREATE MIGRATIONS

5. write the "`replace`" list in the new migration

```
  class Migration(migrations.Migration):

      initial = True

+     replaces = [
+         ("shop", "0001_initial"),
+         ("shop", "0002_customer_is_premium"),
+         ("shop", "0003_shippingaddress"),
+         # ...
+         ("shop", "0026_product"),
+     ]

      dependencies = [
```

RECREATE MIGRATIONS

6. restore old migration files

```
# from command line with something like this
mv -i -v $APP_LABEL/old_migrations/*.py $APP_LABEL/migrations
# check for missing/overwritten files!
```

remove the temporary directory

```
rm -r $APP_LABEL/old_migrations
```

RECREATE MIGRATIONS

7. ensure that old migrations are still there

```
$ manage.py showmigrations $APP_LABEL
shop
 [ ] 0001_init_squashed
 [X] 0001_initial
 [X] 0002_customer_is_premium
 # ...
 [X] 0026_product
```

RECREATE MIGRATIONS

8. launch the migration command

```
$ manage.py migrate $APP_LABEL
Operations to perform:
  Apply all migrations: shop
Running migrations:
  No migrations to apply.
```

ensure that squashed migration has been applied

```
$ manage.py showmigrations $APP_LABEL
shop
 [X] 0001_init_squashed (26 squashed migrations)
```

RECREATE MIGRATIONS

9. back to post-squash tasks

- commit and release
- upgrade all systems with the new release
- remove old migration files, commit and do a second release
- update all migrations that depend on the deleted migrations
- remove the `replaces` attribute
- (optional) prune references to deleted migrations

WHAT COULD POSSIBLY GO WRONG?

MIGRATIONS PROVIDING INITIAL DATA

- create a new migration file for that, after recreating the initial migration

or (even better)

- use fixtures

https://docs.djangoproject.com/en/4.2/howto/initial-data/

CIRCULAR DEPENDENCIES

To manually resolve a CircularDependencyError, break out one of the ForeignKeys in the circular dependency loop into a separate migration, and move the dependency on the other app with it.

If you're unsure, see how makemigrations deals with the problem when asked to create brand new migrations from your models. In a future release of Django, squashmigrations will be updated to attempt to resolve these errors itself.

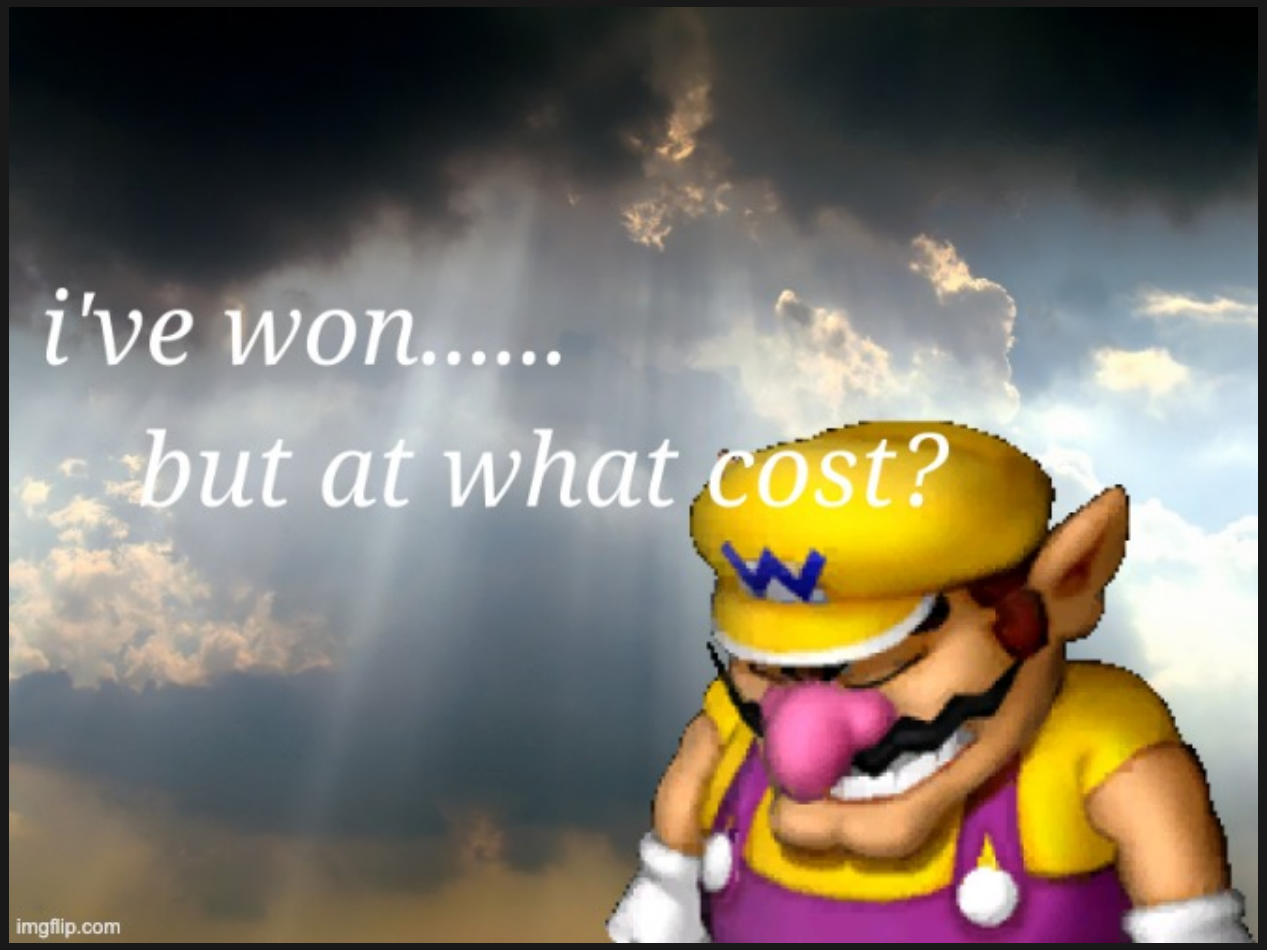
https://docs.djangoproject.com/en/4.2/topics/migrations/#squashing-migrations

TEST PERFORMANCES AFTER RECREATING

```
$ time manage.py test

Ran 152 tests in 0.988s
python manage.py test  5.12s user 0.21s system 82% cpu 6.485 t
```

	Creating test DB	Running tests
Before	~20s	~1s
--keepdb	~0s 🥳	~1s
MIGRATE = False	~25s 😑	~1s
After squashing	~20s 😑	~1s
After recreating	~5s 🥳	~1s


i've won.....
but at what cost?
imgflip.com

THANK YOU!

- github.com/dennybiasiolli/django-squashmigrations-example
  - `django-settings-migrate` branch (PR #4)
  - `squashing-migrations` branch (PR #2)
  - `recreating-migrations` branch (PR #3)

@dennybiasiolli

www.dennybiasiolli.com