GRADUATING FROM DJANGO BEGINNER TO ORM MASTER

Custom Model Managers and QuerySets

JOSH THOMAS

The Westervelt Company

Josh Thomas

- Web Developer at The Westervelt Company
- Used Django professionally for ~3 years
- Django Software Foundation member
- Jazzband member/contributor
- Maintain a couple small Django/Wagtail CMS packages
- BS in Civil Engineering from University of Alabama
- Married with a son (3) and daughter (20 mo)

Website: https://joshthomas.dev

Mastodon: @josh@joshthomas.dev

GitHub: @joshuadavidthomas

We are stewards of the land.

The Westervelt Company

- Founded in 1884
- Headquarters located in Tuscaloosa, AL
  - Southeast US (AL, FL, GA, SC, TN, VA)
  - Northwest US (CO, NE)
  - West US (CA)
  - New Zealand
- Businesses
  - Real Estate
  - Forest Resources
  - Wood Products
  - Ecological Mitigation
  - New Zealand
- Environmental stewardship and resource conservation
- Westervelt 💚 Django

Website: https://westervelt.com

GitHub: @westerveltco

What are Managers?

What are Managers?

“A **Manager** is the interface through which database query operations are provided to Django models.”[1]

- Model methods = row level operations
- Manager methods = table level operations

1. https://docs.djangoproject.com/en/4.2/topics/db/managers/#django.db.models.Manager

```python
from django.db import models


class Question(models.Model):
    question_text = models.CharField(max_length=200)
    pub_date = models.DateTimeField("date published")

    objects = models.Manager()


Question.objects.filter(
    question_text="Is this talk going great?“
)
```

Wait, what?

- If you’ve queried any Model, you’ve used a Manager

```
objects = CustomManager()

objects = CustomQuerySet.as_manager()

objects = CustomManager.from_queryset(CustomQuerySet)()
```

Defining custom Managers

- Manager
- QuerySet
- Manager and QuerySet

Defining custom Managers: Option 1

```python
from django.db import models


class QuestionManager(models.Manager): ...

class Question(models.Model):
    question_text = models.CharField(max_length=200)
    pub_date = models.DateTimeField("date published")

    objects = QuestionManager()
```

- Basic usage

Defining custom Managers: Option 2

```python
from django.db import models


class QuestionQuerySet(models.QuerySet): ...

class Question(models.Model):
    question_text = models.CharField(max_length=200)
    pub_date = models.DateTimeField("date published")

    objects = QuestionQuerySet.as_manager()
```

- Bread and butter usage

Defining custom Managers: Option 3

```python
from django.db import models


class QuestionQuerySet(models.QuerySet): ...

class QuestionManager(models.Manager): ...

class Question(models.Model):
    question_text = models.CharField(max_length=200)
    pub_date = models.DateTimeField("date published")

    objects = QuestionManager.from_queryset(QuestionQuerySet)()
```

- Advanced usage

```python
from django.db import models
from django.utils import timezone


class QuestionManager(models.Manager):
    def published_in_future(self):
        return self.filter(pub_date__gt=timezone.now())

    def questions_about_me(self):
        return self.filter(
            question_text__icontains="Josh Thomas"
        )

class Question(models.Model):
    question_text = models.CharField(max_length=200)
    pub_date = models.DateTimeField("date published")

    objects = QuestionManager()
```

Manager or QuerySet?

- QuerySets = chainable
- Managers = not chainable

```python
# With CustomManager
# Works!
Question.objects.published_in_future()
Question.objects.questions_about_me()

# Does not work!
# Raises AttributeError
Question.objects.published_in_future().questions_about_me()
```

Manager or QuerySet?

- QuerySets = chainable
- Managers = not chainable

```python
from django.db import models
from django.utils import timezone


class QuestionQuerySet(models.QuerySet):
    def published_in_future(self):
        return self.filter(pub_date__gt=timezone.now())

    def questions_about_me(self):
        return self.filter(
            question_text__icontains="Josh Thomas"
        )

class Question(models.Model):
    question_text = models.CharField(max_length=200)
    pub_date = models.DateTimeField("date published")

    objects = QuestionQuerySet.as_manager()
```

Manager or QuerySet?

- QuerySets = chainable
- Managers = not chainable

Manager or QuerySet?

```
# With CustomQuerySet
# Still works!
Question.objects.published_in_future()
Question.objects.questions_about_me()

# But now this does too!
Question.objects.published_in_future().questions_about_me()
```

- QuerySets = chainable
- Managers = not chainable

Manager or QuerySet?

```python
from django.db import models
from django.utils import timezone


class QuestionManager(models.Manager):
    def create_question(self, question: str): ...

class QuestionQuerySet(models.QuerySet):
    def published_in_future(self): ...
    def questions_about_me(self): ...

class Question(models.Model):
    question_text = models.CharField(max_length=200)
    pub_date = models.DateTimeField("date published")

    objects = QuestionManager.from_queryset(QuestionQuerySet)()
```

- QuerySets
  - Filters
  - Annotations/Aggregations
  - Reading
- Managers
  - Creating
  - Updating
  - Deleting

Why use custom Managers?

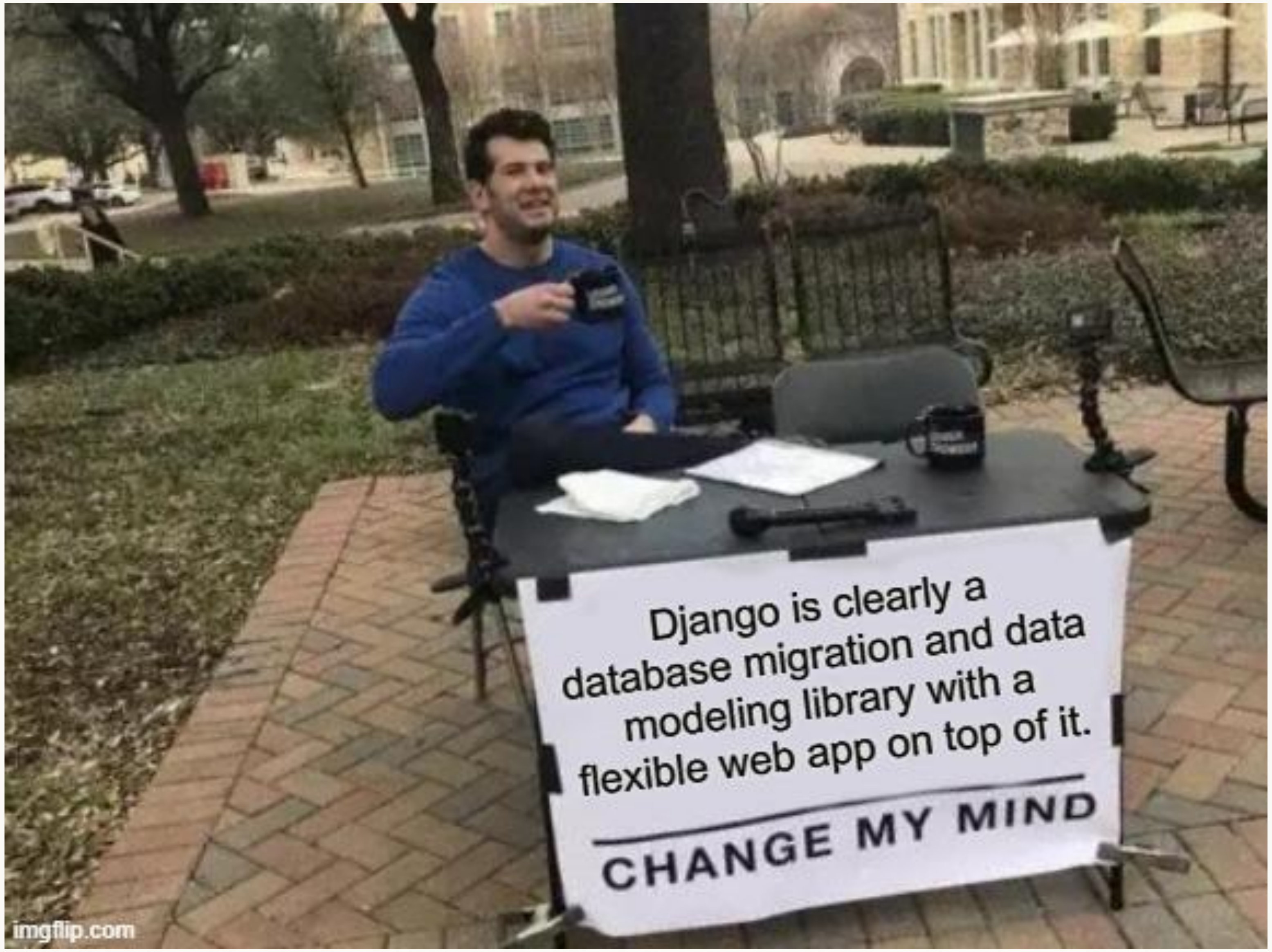
"They are part of the core framework, the documentation is spot-on, they are really useful, and I feel like they are overlooked."[1]

1. Shawn Inman, "Mighty Model Managers", DjangoCon US 2016
   https://www.youtube.com/watch?v=YGwSNkdwAEs

Why use custom Managers?

BECAUSE THEY ARE AWESOME!

- Readability
- Encapsulation
- Testing!!!
- Built-in service layer
- Gateway to ORM


Django is clearly a database migration and data modeling library with a flexible web app on top of it.
CHANGE MY MIND
imgflip.com

1. https://mastodon.social/@webology/111183530617730566
2. https://fosstodon.org/@CodenameTim/111183709866579332

Migrating a legacy application

Migrating a legacy application

- Hunting Club Leases
  - 800 clubs
  - 1200 tracts
- In production since 2006
- Written in ColdFusion

```
SELECT Nvl(SUM(wws_inv_trans.amount), 0) AS Amount_Invoiced
FROM   wws_lease_master,
       wws_lease_detail,
       wws_tract,
       wws_inv_trans,
       wws_lease
       wws_club
WHERE  wws_lease
       AND wws_l
       AND wws_l
       AND wws_l
       AND wws_l
       AND wws_i
       AND wws_i
       AND wws_i
       AND wws_t
           #quot
       AND wws_l
       AND wws_l
       AND wws_l
       AND wws_l
       AND wws_c
       AND wws_c
       AND NOT E
```

```
SELECT club_id
FROM   wws_lease_right,
       wws_lease_right_type
WHERE  wws_lease_right.year = #variables.year#
       AND wws_lease_right.right_id =
le)) =
#
le)) =
#
le)) =
```

```
                        'TURKEY'
                    AND club_id = wws_inv_trans.club_id
             MINUS
```

What did I do?

RELIED HEAVILY ON CUSTOM MODEL MANAGER METHODS, NATCH

- Original tables to Django models
- SQL/CF query to Manager method
  - Descriptively named
  - Original raw SQL as comment
  - Translated to ORM
- Potential fallback to using raw SQL with ORM[1]

1. https://docs.djangoproject.com/en/4.2/topics/db/sql/

Common Patterns

Common Methods I Always Reach For

A COOKBOOK OF CUSTOM MANAGER METHODS, IF YOU WILL

- `.for_CONTEXT(ctx)`
- `.is_CONDITION(cond?)`
- `.not_CONDITION(cond?)`
- `.exclude_CONDITION(cond?)`
- `.create_MODEL()`
- `.create_MODEL_for_CONTEXT(ctx)`
- `.set_FIELD()`
- `.toggle_FIELD()`
- `.within_RANGE(rng)`
- `.greater_than_VALUE(val)`
- `.less_than_VALUE(val)`
- `.with_ANNOTATION()`

.for_CONTEXT(ctx)

```
from django.db import models
from users.models import User


def for_user(self, user: User):
    return self.filter(user=user)
```

- Manager or QuerySet
- Filter based on some context

```python
from django.db import models
from users.models import User


class NewHireQuerySet(models.QuerySet):
    def for_user(self, user: User):
        queryset = self.filter(status="active")
        if not user.is_staff:
            queryset = queryset.filter(
                models.Q(hiring_manager__mail=user.email)
                | models.Q(hiring_manager2__email=user.email)
                | models.Q(old_hiring_manager_email=user.email)
            )
        return queryset
```

.for_CONTEXT(ctx)

- Manager or QuerySet
- Filter based on some context

```python
from django.db import models

from tracts.constants import NO_HUNTING_ACCESS_DEER_TRAX_IDS


class TractQuerySet(models.QuerySet):
    def is_leasable(self):
        return self.filter(
            status="active"
        ).exclude_no_hunting_access()

    def exclude_no_hunting_access(self):
        return self.exclude(
            deer_trax_id__in=NO_HUNTING_ACCESS_DEER_TRAX_IDS
        )

    def is_leased_for_year(self, year: int):
        return self.is_leasable().filter(lease__year=year)
```

.is_CONDITION(cond?)
.not_CONDITION(cond?)
.exclude_CONDITION(cond?)

- Manager or QuerySet
- Filter based on some condition

```python
from django.contrib.auth.models import Group
from django.contrib.auth.models import UserManager

from clubs.models import Club


class CustomUserManager(UserManager):
    def create_user_for_club(self, club: Club, **kwargs):
        user = self.model.objects.create_user(
           username=str(club.id),
           email=club.officer.email if club.officer else "",
           first_name=club.name,
           **kwargs,
        )
        user.groups.add(
           Group.objects.get(name="Club Users")
        )
        return user
```

.create_MODEL()
.create_MODEL_for_CONTEXT(ctx)

- Manager
- Create a model with side effects

```python
import datetime
from django.db import models
from django.utils import timezone


class NewHireSurveyQuerySet(models.QuerySet):
    def within_start_date(self, days: int):
        return self.filter(
            new_hire__start_date__gte=(
                timezone.localtime()
                - datetime.timedelta(days=days)
            ).date(),
            new_hire__start_date__lte=timezone.localtime().date(),
        )
```

.within_RANGE()

- Manager or QuerySet
- Filter based on numerical/date range

```python
import datetime
from django.db import models
from django.utils import timezone


class NewHireSurveyQuerySet(models.QuerySet):
    def greater_than_notification_days(
        self, days: int
    ):
        return self.annotate(
            last_notified_at_within_days=models.Case(
                models.When(
                   last_notified_at__gte=(
                       timezone.localtime()
                       - datetime.timedelta(days=days)
                   ).date(),
                   last_notified_at__lte=timezone.localtime().date(),
                   then=models.Value(True),
                ),
                default=models.Value(False),
            )
        ).filter(
            last_notified_at_within_days=False
        )
```

.greater_than_VALUE()
.less_than_VALUE()

- Manager or QuerySet
- Filter based on numerical/date value

```python
from django.db import models


def set_active(self):
    self.update(status="active")

def toggle_status(self):
    self.update(
        status=models.Case(
            models.When(status="active", then="inactive"),
            models.When(status="inactive", then="active"),
            default="active",
            output_field=models.CharField(),
        )
    )
```

.set_FIELD()
.toggle_FIELD()

- Manager or QuerySet
- Update a field across entire QuerySet

```python
from django.db import models
from adjustments.models import Adjustment
from adjustments.models import AdjustmentType

class ClubTractQuerySet(models.QuerySet):
    def with_adjusted_acres(self):
        queryset = Adjustment.objects.filter(
            tract=models.OuterRef("tract__id"),
            year=models.OuterRef("year"),
            adjustment_type=AdjustmentType.ACRES,   # “A”
        ).values("tract__id", "year")

        subquery = queryset.annotate(
            sum=models.Sum("amount")
        ).values("sum")

        return self.annotate(
            acres_adjustments=models.Subquery(subquery)
        ).annotate(
            adjusted_acres=models.Case(
                models.When(
                    models.Q(
                        acres_adjustments__isnull=False
                    ),
                    then=models.F("acres")
                    + models.F("acres_adjustments"),
                ),
                default=models.F("acres"),
                output_field=models.FloatField(),
            ),
        )
```

.with_ANNOTATION()

- Manager or QuerySet
- Add additional info not available on Model

```python
from invoices.models import Invoice
from system.constants import CURRENT_YEAR


def with_total_invoiced_amount(
    self, year: int = CURRENT_YEAR
):
    subquery = (
        Invoice.objects.for_mgmt_unit_in_year(
            models.OuterRef("pk"), year
        )
        .filter(legacy_id__startswith="361L")
        .values(
           "club__lease_details__tract__management_unit__pk"
        )
        .annotate(
           amount_invoiced=models.Sum(
               "amount", distinct=True
           )
        )
        .values("amount_invoiced")
    )

    return self.annotate(
        total_invoiced_amount=models.Subquery(subquery),
    )
```

.with_ANNOTATION()

- Manager or QuerySet
- Add additional info not available on Model

Things missing

HAD TO FIT THIS TALK IN 25 MINUTES SOMEHOW

- Modifying the default QuerySet
- Renaming default Manager and defining multiple Managers
- Usage with Abstract Models
- `.select_related()` or `.prefetch_related()`

References

- Django documentation
  - https://docs.djangoproject.com/en/4.2/topics/db/managers/
  - https://docs.djangoproject.com/en/4.2/topics/db/sql/
- "Mighty Model Managers" – Shawn Inman, DjangoCon US 2016
  - https://www.youtube.com/watch?v=YGwSNkdwAEs
  - https://2016.djangocon.us/schedule/presentation/62/
- https://mastodon.social/@webology/111183530617730566
- https://fosstodon.org/@CodenameTim/111183709866579332

Slides

https://github.com/joshuadavidthomas/dcus-2023-talk

Thanks

- Django creators, contributors, educators
- DjangoCon US 2023 organizers and volunteers
- Django community
- My family
- You