Tips for Optimizing Django Database Queries

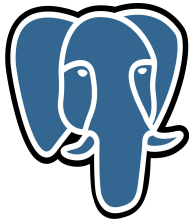
Albert Fernández Bufias
Sergio González Cruz

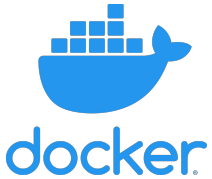
Workshop requirements

https://github.com/paradoxa-tech/django-optimization-tips

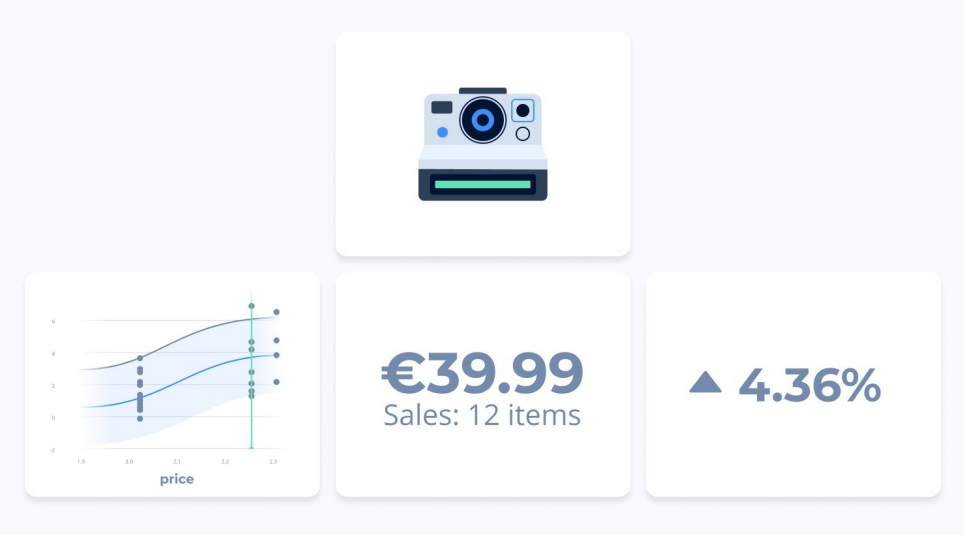
Follow the README instructions

About us

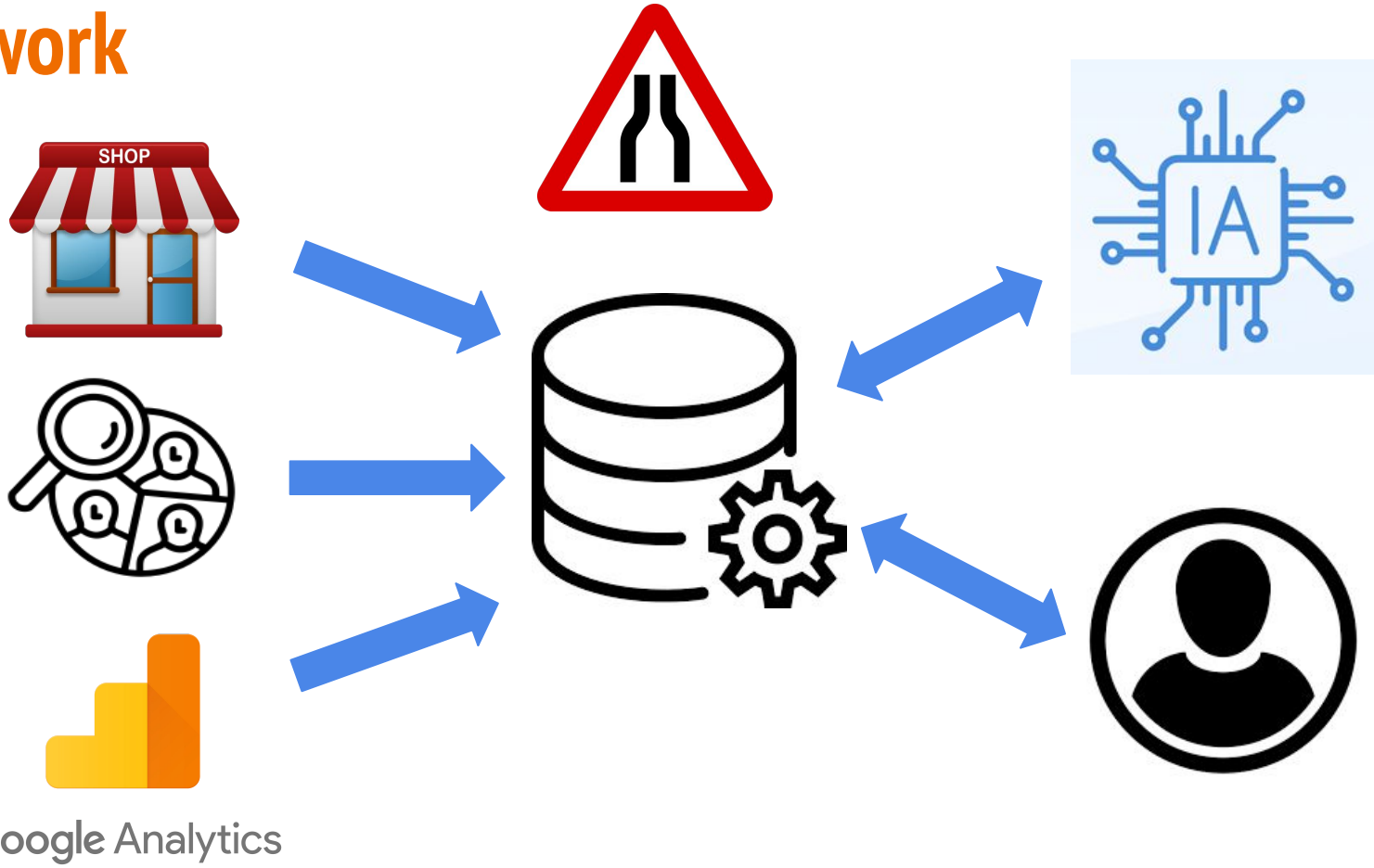




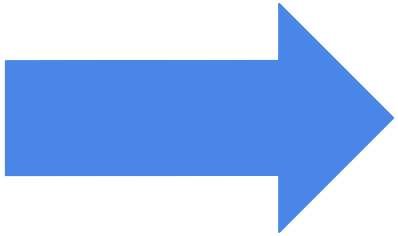
optimus price



Our work



The problem and solution



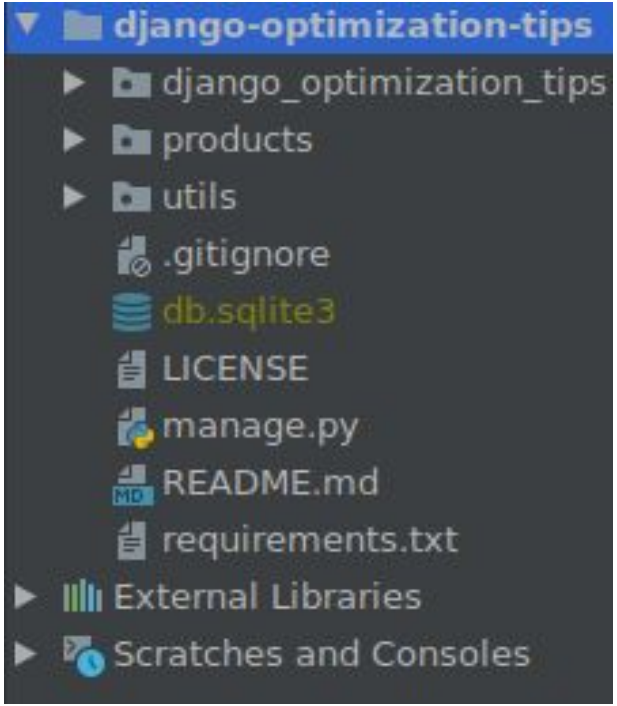
The project

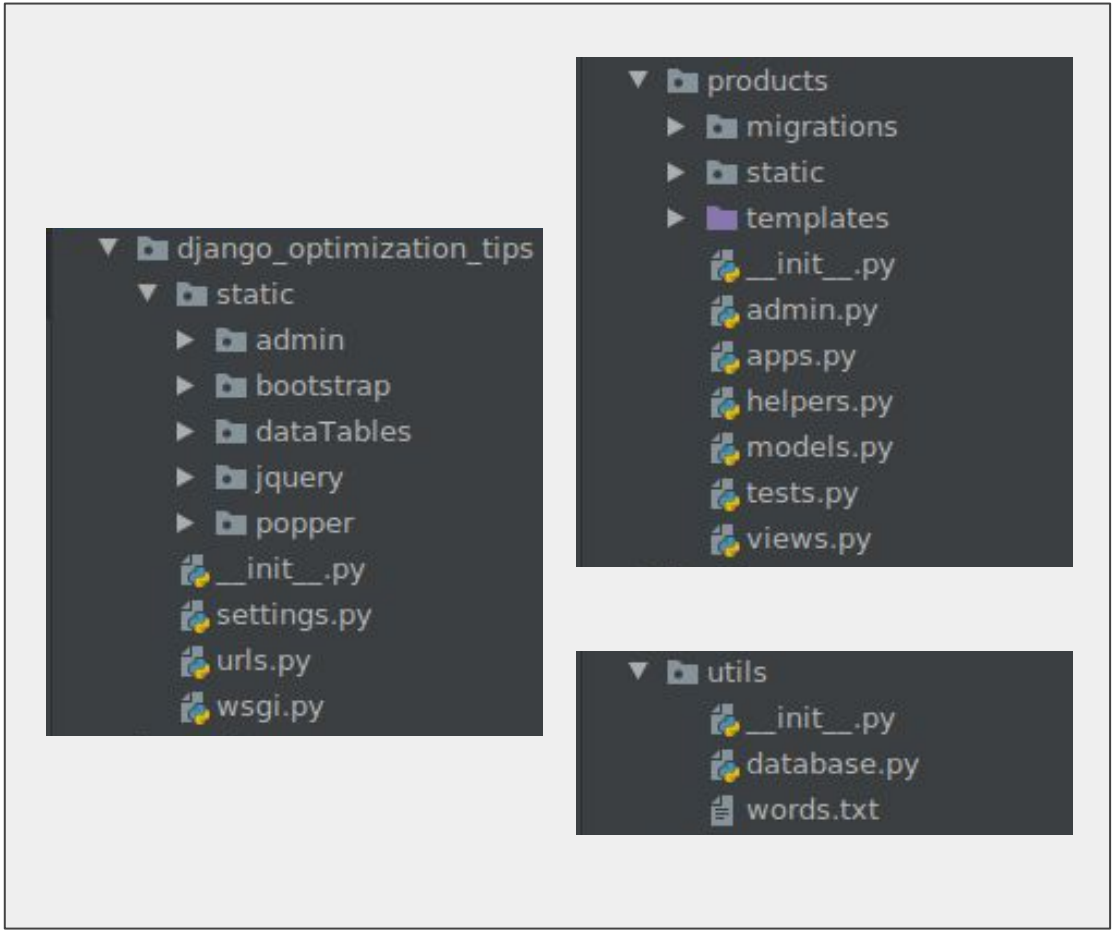
We have created a little database with random data, simulating the database of a shop (but in a much more simple way).

We have added Django on top of it to be able to grow quickly the application and because is an awesome ORM. The problems start when we have to work with big bunches of data.

The objective of the workshop is to optimize the existing code using Django tools and some strategies that we have learned, in order to be able to have the required data as fast as possible from the backend (as good backend developers we want to reduce as much as possible the frontend work).

The project tree



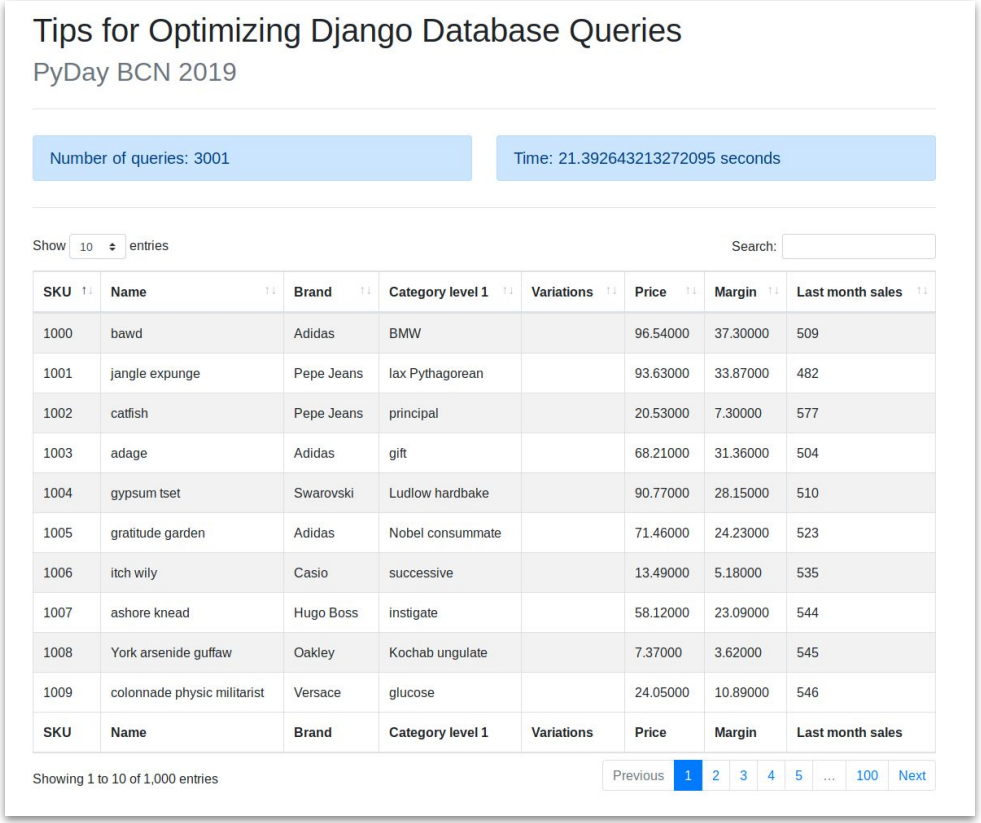


Run the project

```
$ source dot_venv/bin/activate

$ python manage.py runserver 9000

In the browser:
http://localhost:9000
```



Tips for Optimizing Django Database Queries

PyDay BCN 2019

Number of queries: 3001

Time: 21.392643213272095 seconds

Show 10 entries

Search:

SKU	Name	Brand	Category level 1	Variations	Price	Margin	Last month sales
1000	bawd	Adidas	BMW		96.54000	37.30000	509
1001	jangle expunge	Pepe Jeans	lax Pythagorean		93.63000	33.87000	482
1002	catfish	Pepe Jeans	principal		20.53000	7.30000	577
1003	adage	Adidas	gift		68.21000	31.36000	504
1004	gypsum tset	Swarovski	Ludlow hardbake		90.77000	28.15000	510
1005	gratitude garden	Adidas	Nobel consummate		71.46000	24.23000	523
1006	itch wily	Casio	successive		13.49000	5.18000	535
1007	ashore knead	Hugo Boss	instigate		58.12000	23.09000	544
1008	York arsenide guffaw	Oakley	Kochab ungulate		7.37000	3.62000	545
1009	colonnade physic militarist	Versace	glucose		24.05000	10.89000	546
SKU	Name	Brand	Category level 1	Variations	Price	Margin	Last month sales

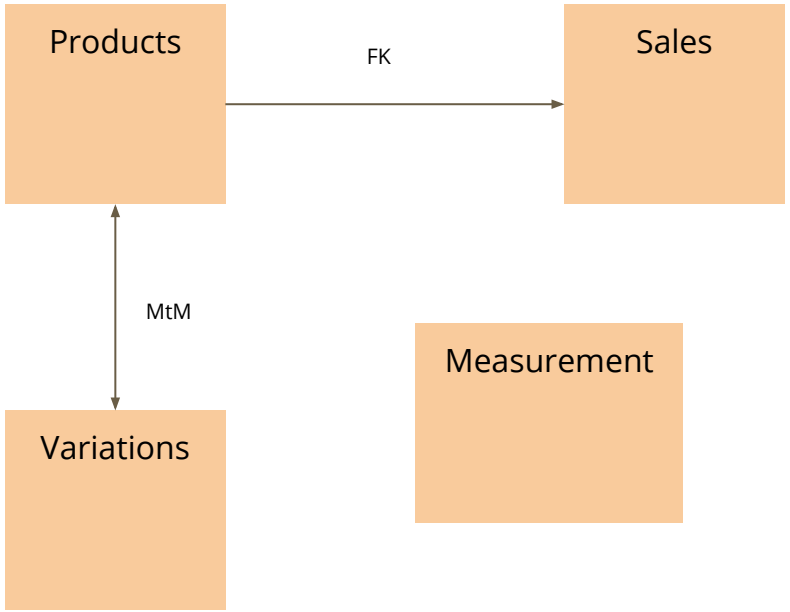
Showing 1 to 10 of 1,000 entries

Previous 1 2 3 4 5 … 100 Next

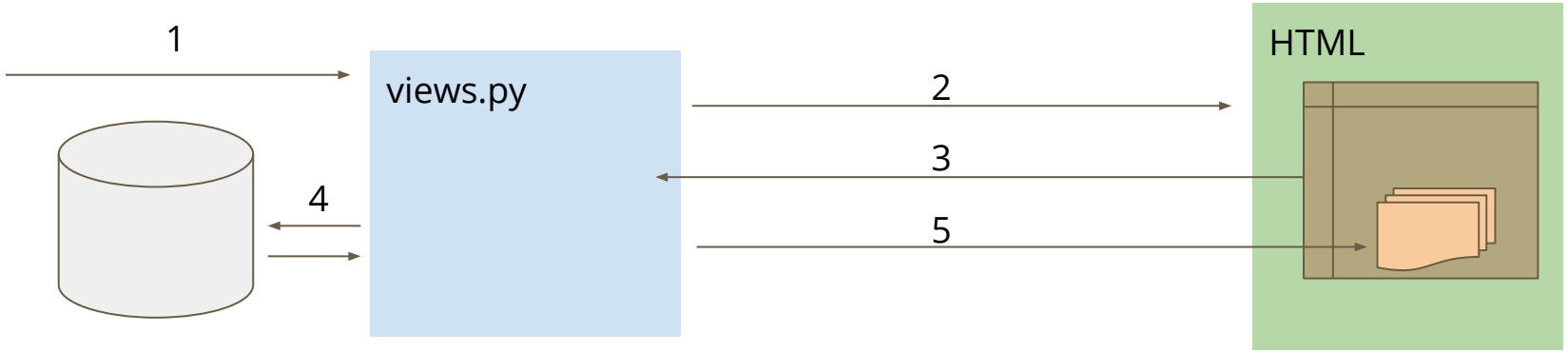
The models

The database has 3 tables:

- Product (shop products)
- Sale (the sales of all products)
- Variation (the variations of the products such as color, size, etc.).



The views



1. User request
2. View provides the HTML and the table skeleton
3. The table callback go to the callback view -> **ProductTableView**
4. The callback view provides the body (data) -> **ProductTableCallbackView**

How to start?

- Identify the bottlenecks
  - Django Debug Toolbar
    https://django-debug-toolbar.readthedocs.io/en/latest/
  - django.db conexion
    https://docs.djangoproject.com/en/2.2/topics/db/sql/
  - Time
    https://docs.python.org/3.6/library/time.html
  - explain() (New in Django 2.1)
    https://docs.djangoproject.com/en/2.2/ref/models/querysets/#django.db.models.query.QuerySet.explain

Tip 1: Laziness and Caching

- QuerySets are lazy
  https://docs.djangoproject.com/en/2.2/topics/db/queries/#querysets-are-lazy

- QuerySets are cached
  https://docs.djangoproject.com/en/2.2/topics/db/queries/#caching-and-querysets

Task 1: Laziness and Caching (I)

```
django-optimization-tips$ python manage.py shell

>>> from django.db import connection
>>> from products.models import Product

>>> initial_number_of_queries = len(connection.queries)
>>> prods_0 = Product.objects.filter(ean=1)
>>> prods_1 = Product.objects.filter(part_number=3)
>>> prods_2 = Product.objects.filter(cost__gt=4).filter(brand="Klaatu")
>>> len(connection.queries) - initial_number_of_queries
0

>>> products = prods_2.values() # Evaluate here
>>> len(connection.queries) - initial_number_of_queries
1
```

Task 1: Laziness and Caching (II)

```
>>> import time

>>> from products.models import Sale

>>> def caching():
...     t1 = time.time()
...     list(Sale.objects.all().values()) # Hit to database
...     t2 = time.time()
...     list(Sale.objects.all().values()) # From cache
...     t3 = time.time()
...     print(f"{t2 - t1}")
...     print(f"{t3 - t2}")
...
>>> caching()

2.90224432945251 46
1.8833801746368408
```

Tip 2: Select Related and Prefetch Related

Django provides us with these 2 tools:

- **select_related**: Foreign key relationship
  https://docs.djangoproject.com/en/2.2/ref/models/querysets/#select-related
- **prefetch_related**: Many To Many relationship
  https://docs.djangoproject.com/en/2.2/ref/models/querysets/#prefetch-related

```
# DON'T
sale = Sale.objects.get(id=5)  # Hits the database.
product = sale.product  # Hits the database again

# DO
sale = Sale.objects.select_related('product').get(id=5) # Hits the
database.
product = sale.product  # Doesn't hit the database.
```

Tip 3: Get only what you need

- values (“field1”, “field2”, ...) -> dictionaries
  https://docs.djangoproject.com/en/2.2/ref/models/querysets/#values

- values_list (“field1”, “field2”, ...) -> tuples
  https://docs.djangoproject.com/en/2.2/ref/models/querysets/#values-list

- defer (“field1”, “field2”, ...) -> objects
  https://docs.djangoproject.com/en/2.2/ref/models/querysets/#defer

- only (“field1”, “field2”, ...) -> objects
  https://docs.djangoproject.com/en/2.2/ref/models/querysets/#only

Task 2: Problem

- **Optimize the query using tips 2 and 3**

  In products/helpers.py:
  def task_2_generate_data(self)

Help

- only
- prefetch_related

Task 2: Solution

```
def task_2_generate_data(self):
    table_info = []

    products = Product.objects.all().only(
        'sku', 'name', 'product_variations', 'brand',
        'category_level_1', 'current_price', 'cost'
    ).prefetch_related('product_variations')

    for product in products:
        product_info = dict()

        ...
```

Task 2: Github

```
git fetch


git checkout task2

or

git checkout origin/task2
```

Tip 4: Count and Exists

- Use count() and exists() when you don’t need the contents of the QuerySet

```
# DON'T
count = len(Product.objects.all())   # Evaluates the entire queryset

# DO
count = Product.objects.all().count()   # Executes more efficient SQL
```

```
# DON'T
exists = len(Product.objects.filter(name='Iphone XX')) > 0

# DO
exists = Product.objects.filter(name='Iphone XX').exists()
```

Tip 5: Take care with the loops

```
python manage.py shell

>>> from utils.tip_5 import
looping
>>> looping()
```

```python
def looping():
    # DON'T
    products = Product.objects.filter(id__lte=5)
    products_with_low_id = {}
    for id_ in range(1, 5):
        try:
            # Database query on each iteration
            products_with_low_id[id_] = products.get(id=id_)
        except Product.DoesNotExist:
            pass
    # DO
    products = Product.objects.filter(id__lte=5)
    products_with_low_id = {}
    # Evaluate the QuerySet and construct lookup
    lookup = {product.id: product for product in products}
    for id_ in range(1, 5):
        try:
            # No database query
            products_with_low_id[id_] = lookup[id_]
        except KeyError:
            pass
```

Tip 6: Let database work (I)

- Use filter() and exclude() for filtering: https://docs.djangoproject.com/en/2.2/topics/db/queries/#retrieving-specific-objects-with-filters

```
# DON'T
for product in Product.objects.all():
    if "pyday" in product.name.lower():
        # Do something

# DO
for person in Product.objects.filter(name__icontains="pyday"):
    # Do something
```

Tip 6: Let database work (II)

- Use Q expressions: https://docs.djangoproject.com/en/2.2/topics/db/queries/#complex-lookups-with-q-objects

```
# DON'T
adidas_products = list(Product.objects.filter(brand="adidas"))
nike_products = list(Product.objects.filter(brand="nike"))
adidas_or_nike_products = list(set(adidas_products + nike_products))

# DO
adidas_or_nike_products = \
    list(Product.objects.filter(Q(brand='adidas') | Q(brand='nike')))
```

Tip 6: Let database work (III)

- Use aggregation: https://docs.djangoproject.com/en/2.2/topics/db/aggregation/

```
# DON'T
total_stock = 0
for product in Product.objects.all():
    total_stock += product.stock

# DO
from django.db.models import Sum
total_stock = Product.objects.all().aggregate(
    Sum('stock')
)['stock__sum']
```

Task 3: Problem

- **Let the database work to get the sales of the last month:**

  def task_3_get_last_month_sales (product)

Help

- Aggregate
- Sum

Task 3: Solution

```
from django.db.models import Sum

def task_3_get_last_month_sales(product):
    today = datetime.date.today()
    last_month_date = today - datetime.timedelta(days=30)
    return Sale.objects.filter(
        product=product, date__gte=last_month_date
    ).aggregate(
        Sum('quantity_purchased')
    )['quantity_purchased__sum']
```

Task 3: Github

```
git fetch


git checkout task3

or

git checkout origin/task3
```

Task 4: Problem

- **Let the database work to search in the table:**

def task_4_search_data (self)

Help:

- def task_4_search_data (self, products_queryset)
- helpers.py: task_2_generate_data()
- views.py: ProductTablaView
- filter: __icontains

Task 4: Solution (I)

```
from django.db.models import Q

def task_4_search_data(self, products_queryset):
    if not self.search:
        return products_queryset

    return products_queryset.filter(
        Q(sku__icontains=self.search) |
        Q(name__icontains=self.search) |
        Q(brand__icontains=self.search) |
        Q(category_level_1__icontains=self.search)
    )
```

Task 4: Solution (II)

```
def task_2_generate_data(self):

    table_info = []
    products = Product.objects.all().only(
        'sku', 'name', 'product_variations', 'brand',
        'category_level_1', 'current_price', 'cost'
    ).prefetch_related('product_variations')

    products = self.task_4_search_data(products)

    for product in products:
        ...
```

Task 4: Solution (III)

```
class ProductTableView(View):

    ...

    # process_table_data.task_4_search_data()
    process_table_data.task_5_sort_data()
    data = process_table_data.get_data()

    ...
```

Task 4: Github

```
git fetch


git checkout task4

or

git checkout origin/task4
```

Task 5: Problem

- **Let the database work to sort the table:**

task_5_sort_data

Help:

- helpers.py: task_5_sort_data (self, products_queryset)
- helpers.py: task_2_generate_data ()
- views.py: ProductTablaView
- order_by

Task 5: Solution (I)

```
def task_5_sort_data(self, products_queryset):

    headers_index = {
        "0": "sku",
        "1": "name",
        "2": "brand",
        "3": "category_level_1",
        "5": "price",
        "6": "margin",
        "7": "last_month_sales"
    }

    column = headers_index[self.order_column]
    if self.order_dir == "asc":
        column = '-' + column
    return products_queryset.order_by(column)
```

Task 5: Solution (II)

```
def task_2_generate_data(self):

    table_info = []
    products = Product.objects.all().only(
        'sku', 'name', 'product_variations', 'brand',
        'category_level_1', 'current_price', 'cost'
    ).prefetch_related('product_variations')

    products = self.task_4_search_data(products)
    sorted_products = self.task_5_sort_data(products)

    for product in sorted_products:
        ...
```

Task 5: Solution (III)

```
class ProductTableView(View):

    ...

    # process_table_data.task_4_search_data()
    # process_table_data.task_5_sort_data()

    data = process_table_data.get_data()

    ...
```

Task 5: Github

```
git fetch


git checkout task5

or

git checkout origin/task5
```

Tip 7: Bulks

- Bulk create
  https://docs.djangoproject.com/en/2.2/ref/models/querysets/#bulk-create

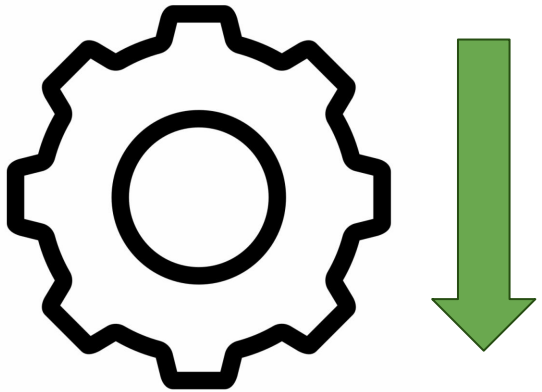
- Bulk update (version 2.2)
  https://docs.djangoproject.com/en/2.2/ref/models/querysets/#bulk-update

```
products = []

for index in range(10):
    product = Product(name=str(index0))
    products.append(product)

Product.objects.bulk_create(products)
```

Tip 8: Intermediate table



Task 6: Problem

- **Create an intermediate table:**

1. Create a new model IntermediateProduct with only the columns of the table

2. Create a function in helpers “update_intermediate_products”:
   - Delete all the IntermediateProduct
   - Generate the IntermetiateProduct with the products
   - Execute it using the shell for fill the new database model.

3. Use task_2_generate_data to use the IntermediateProduct for showing the data

Task 6: Solution (I)

```
class IntermediateProduct(models.Model):

    product = models.ForeignKey(Product , on_delete=models.CASCADE)
    sku = models.CharField( max_length=1000, blank=True, null=True)
    name = models.CharField( max_length=1000, blank=True, null=True)
    brand = models.CharField( max_length=1000, blank=True, null=True)
    category_level_1 = models.CharField( max_length=1000, blank=True,
null=True)
    variations = models.CharField( max_length=1000, blank=True, null=True)
    price = models.FloatField()
    margin = models.FloatField()
    last_month_sales = models.IntegerField()
```

```
$ python manage.py makemigrations products

$ python manage.py migrate
```

Task 6: Solution (II)

```
def update_intermediate_products():
    intermediate_products = []
    products = Product.objects.all().prefetch_related("product_variations")
    for product in products:
        intermediate_product = IntermediateProduct()
        intermediate_product.product = product
        intermediate_product.sku = product.sku
        intermediate_product.name = product.name
        intermediate_product.brand = product.brand
        intermediate_product.category_level_1 = product.category_level_1
        intermediate_product.price = product.current_price
        intermediate_product.variations = product.product_variations
        intermediate_product.profit = ProcessTableData.get_product_profit(product)
        intermediate_product.last_month_sales = \
            ProcessTableData.task_3_get_last_month_sales(product)

        intermediate_products.append(intermediate_table_row)

    IntermediateProduct.objects.all().delete()
    IntermediateProduct.objects.bulk_create(intermediate_products)
```

Task 6: Solution (III)

```
$ python manage.py shell

>>> from products.helpers import update_intermediate_products

>>> update_intermediate_products()
```

Task 6: Solution (IV)

```
def task_2_generate_data(self):
    table_info = []
    products = IntermediateTable.objects.all()
    products = self.task_4_search_data(products)
    sorted_products = self.task_5_sort_data(products)

    for product in sorted_products:
        product_info = dict()
        product_info["sku"] = product.sku
        product_info["name"] = product.name
        product_info["variations"] = product.variations
        product_info["brand"] = product.brand
        product_info["category_level_1"] = product.category_level_1
        product_info["profit"] = product.profit
        product_info["price"] = product.price
        product_info["last_month_sales"] = product.last_month_sales
        table_info.append(product_info)

    return table_info
```

Task 6: Github

```
git fetch


git checkout task6

or

git checkout origin/task6
```

Extra

- Iterator()
  https://docs.djangoproject.com/en/2.2/ref/models/querysets/#iterator

- Index
  https://docs.djangoproject.com/en/2.2/topics/db/optimization/#retrieve-individual-objects-using-a-unique-indexed-column

- Raw Sql:
  https://docs.djangoproject.com/en/2.2/topics/db/optimization/#use-raw-sql

Extra

- Use F expressions: https://docs.djangoproject.com/en/2.2/ref/models/expressions/#f-expressions

Recommended links

- https://docs.djangoproject.com/en/2.2/topics/db/optimization/
- http://concisecoder.io/2018/11/04/django-orm-optimization-tips/
- https://medium.com/@ryleysill93/basic-performance-optimization-in-django-ebd19089a33f
- https://medium.com/@hansonkd/performance-problems-in-the-django-orm-1f62b3d04785
- https://simpleisbetterthancomplex.com/tips/2016/05/16/django-tip-3-optimize-database-queries.html

Questions?

- Sergio González Cruz: gonzalezcruz.sergio@gmail.com

- Albert Fernández Bufias: albert.fernandez.bufias@gmail.com