

Queries performáticas com ORM em Python, Django e Postgres

Thiago Ferreira / Python Brasil 2022

Sobre mim

🤖 Staff Software Engineer @ Ratable

🐍 Pythonista

🌱 Entusiasta da cozinha (Insta: @maisumdiarioveg)

🐶 Pai de pet

🍿 Mandem recomendações de filmes, séries e livros :P

Github: @thiagoferreiraw

Twitter e Medium: @tferreiraw

Linkedin: <https://www.linkedin.com/in/thiago-ferreira-380427a8/>



Sumário



Implementando queries complexas no ORM



Erros comuns nas queries (N+1)



Problemas complexos além do ORM: **índices**



Menções honrosas: Ferramentas e dicas para auxiliar na organização e manutenção do ORM/queries



Conhecimento prático e com muitos exemplos



Chico

Desenvolvedor de software na empresa Kats





Jorge

Chefe na empresa Kats

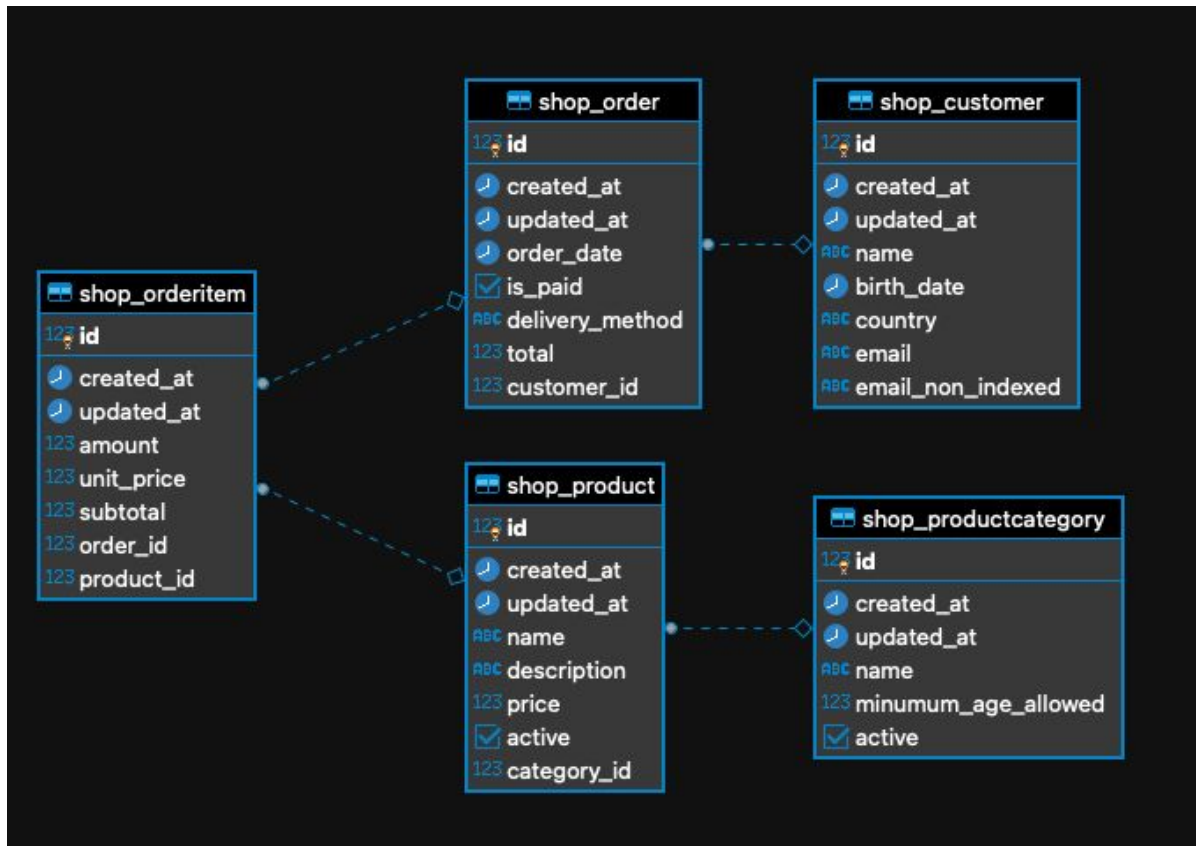
Chico, boas-vindas ao time!

**Sua primeira tarefa será criar
uma listagem de pedidos,
exibindo ID, Nome e total
vendido.**

Tranquilo?

```
Order #481 - Savannah Johnson - $1.39
Order #3793 - Savannah Johnson - $103.23
Order #56 - Sandra Barrett PhD - $263.64
Order #2070 - Sandra Barrett PhD - $143.29
Order #3439 - Sandra Barrett PhD - $183.31
```

Modelagem do banco de dados



Listagem de pedidos



Primeira tarefa: implementar uma listagem de pedidos, exibindo ID, Nome do cliente e total.

```
1 @debug_queries()
2 def list_orders_bad(limit=5):
3     orders = Order.objects.filter()[0:limit]
4     for order in orders:
5         print(f"Order #{order.id} - {order.customer.name} - ${order.total}")
```


Hora do deploy!






Jorge

Chefe na empresa Kats

 Chico, os clientes estão reclamando que a listagem de pedidos está lenta.

Poderia verificar?



Será que o
ORM é
lento?

Banco
sobrecarregado?

É lentidão do
python?

Já estraguei o
brinquedo no
primeiro dia de
serviço 😞

Identificando problemas no ORM em 3 passos:

- 1 Visualizar as queries executadas
- 2 Analisar o comando SQL gerado pelo ORM
- 3 Otimizar de acordo com os resultados - sempre testando antes e depois.

6 queries executadas



```
-----  
[6] SQL statements executed (Total time = 13.8ms, SQL time = 11.0ms):
```

```
1 FROM "shop_order" LIMIT 5
```

```
2 FROM "shop_customer" WHERE "shop_customer"."id" = 511 LIMIT 21
```

```
3 FROM "shop_customer" WHERE "shop_customer"."id" = 412 LIMIT 21
```

```
4 FROM "shop_customer" WHERE "shop_customer"."id" = 989 LIMIT 21
```

```
5 FROM "shop_customer" WHERE "shop_customer"."id" = 866 LIMIT 21
```

```
6 FROM "shop_customer" WHERE "shop_customer"."id" = 572 LIMIT 21  
-----
```

Por que tantas queries?

```
1 @debug_queries()
2 def list_orders_bad(limit=5):
3     orders = Order.objects.filter()[:limit]
4     for order in orders:
5         print(f"Order #{order.id} - {order.customer.name} - ${order.total}")
```

Buscando apenas orders
(sem customer)



O customer é usado ao longo do programa, isso faz com que o Django busque os objetos sob demanda

Queries extras em customer

[6] SQL statements executed (Total time = 13.8ms, SQL time = 11.0ms):

1 FROM "shop_order" LIMIT 5

**Uma nova query para cada iteração
da lista de orders (ids diferentes)**

2 FROM "shop_customer" WHERE "shop_customer"."id" = 511 LIMIT 21

3 FROM "shop_customer" WHERE "shop_customer"."id" = 412 LIMIT 21

4 FROM "shop_customer" WHERE "shop_customer"."id" = 989 LIMIT 21

5 FROM "shop_customer" WHERE "shop_customer"."id" = 866 LIMIT 21

6 FROM "shop_customer" WHERE "shop_customer"."id" = 572 LIMIT 21

Queries N+1

É um problema de performance, onde para cada iteração do laço principal (ex: Pedidos) é executada uma nova query para buscar dados adicionais (ex: Clientes)

Então se temos 10 pedidos na consulta, teremos 1 query para os pedidos e + 10 queries extras para trazer os clientes

Como resolver?

```
1 @debug_queries()
2 def list_orders_good(limit=5):
3     orders = Order.objects.filter().select_related("customer")[:limit]
4     for order in orders:
5         print(f"Order #{order.id} - {order.customer.name} - ${order.total}")
```

Informamos ao ORM para já carregar o customer do banco de dados

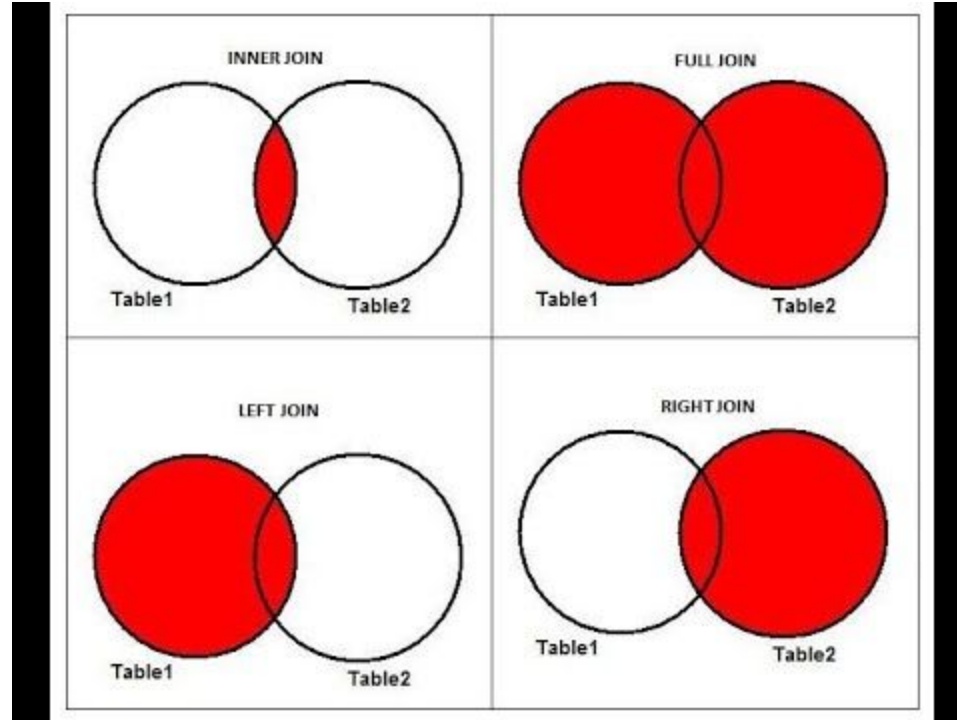
De 6 para 1 query

76% mais rápido que o anterior

```
-----  
[1] SQL statements executed (Total time = 3.2ms, SQL time = 2.0ms)
```

```
1 FROM "shop_order" INNER JOIN "shop_customer" ON ("shop_order"."customer_id" =  
"shop_customer"."id") LIMIT 5  
-----
```

SQL joins e a teoria dos conjuntos



#AgoraVai!



Comparação com SQL puro

```
1 @debug_queries()
2 def list_orders_without_orm(limit=5):
3     with connection.cursor() as cursor:
4         cursor.execute(
5             'select "shop_order"."id", '
6             ' "shop_customer"."name", '
7             ' "shop_order"."total" '
8             'FROM "shop_order" '
9             'INNER JOIN "shop_customer" '
10            ' ON ("shop_order"."customer_id" = "shop_customer"."id") '
11            "LIMIT 5"
12
13        )
14        orders = cursor.fetchall()
15
16        for (order_id, customer_name, total) in orders:
17            print(f"Order #{order_id} - {customer_name} - ${total}")
```

Sem diferenças relevantes em relação ao ORM

Melhoria de 0.6ms 😊 ~18% em relação ao ORM otimizado. Nada muito impressionante.

```
In [64]: list_orders_without_orm()
Order #481 - Savannah Johnson - $1.39
Order #3793 - Savannah Johnson - $103.23
Order #56 - Sandra Barrett PhD - $263.64
Order #2070 - Sandra Barrett PhD - $143.29
Order #3439 - Sandra Barrett PhD - $183.31
```

```
-----
[1] SQL statements executed (Total time = 2.6ms, SQL time = 2.0ms):
```

```
1 FROM "shop_order" INNER JOIN "shop_customer" ON ("shop_order"."customer_id"
p_customer"."id") LIMIT 5
-----
```

Complicando um pouco - 1-N



Segunda tarefa: Listar um pedido com seus respectivos itens
(exibir nome do produto e categoria)

```
1 @debug_queries()
2 def list_order_items_bad(order_id=1):
3     order = Order.objects.get(id=order_id)
4     print(f"Printing Order #{order.id} - {order.customer.name}", "\n", "-" * 60)
5     for item in order.items.all():
6         print("Product: ", item.product.name)
7         print("Category: " item.product.category.name)
8         print("Subtotal: ", item.subtotal, "\n")
```

Muitas queries executadas (11)



[11] SQL statements executed (Total time = 24.2ms, SQL time = 15.0ms):

```
1 FROM "shop_order" WHERE "shop_order"."id" = 1 LIMIT 21
2 FROM "shop_customer" WHERE "shop_customer"."id" = 510 LIMIT 21
3 FROM "shop_orderitem" WHERE "shop_orderitem"."order_id" = 1
4 FROM "shop_product" WHERE "shop_product"."id" = 627 LIMIT 21
5 FROM "shop_productcategory" WHERE "shop_productcategory"."id" = 12 LIMIT 21
6 FROM "shop_product" WHERE "shop_product"."id" = 53 LIMIT 21
7 FROM "shop_productcategory" WHERE "shop_productcategory"."id" = 1 LIMIT 21
8 FROM "shop_product" WHERE "shop_product"."id" = 336 LIMIT 21
9 FROM "shop_productcategory" WHERE "shop_productcategory"."id" = 1 LIMIT 21
10 FROM "shop_product" WHERE "shop_product"."id" = 700 LIMIT 21
11 FROM "shop_productcategory" WHERE "shop_productcategory"."id" = 11 LIMIT 21
```

Resolvendo:

```
1 @debug_queries()
2 def list_order_items_good(order_id=1):
3     order = Order.objects.select_related("customer").get(id=order_id)
4     print(f"Printing Order #{order.id} - {order.customer.name}", "\n", "-" * 60)
5     for item in order.items.all().select_related("product__category"):
6         print("Product: ", item.product.name)
7         print("Category: ", item.product.category.name)
8         print("Subtotal: ", item.subtotal, "\n")
```


De 11 para 2 queries

73% mais rápido que o anterior

[2] SQL statements executed (Total time = 6.4ms, SQL time = 4.0ms):

```
1 FROM "shop_order" INNER JOIN "shop_customer" ON ("shop_order"."customer_id" =  
"shop_customer"."id") WHERE "shop_order"."id" = 1 LIMIT 21
```

```
2 FROM "shop_orderitem" INNER JOIN "shop_product" ON ("shop_orderitem"."product_  
id" = "shop_product"."id") INNER JOIN "shop_productcategory" ON ("shop_product".  
"category_id" = "shop_productcategory"."id") WHERE "shop_orderitem"."order_id" =  
1
```

Complicando MAIS: prefetch



Terceira tarefa: Listar vários pedidos com seus respectivos itens (exibir nome do produto e categoria)

```
1 @debug_queries()
2 def list_multiple_orders_and_items_bad(limit=5):
3     orders = Order.objects.filter()[limit]
4     for order in orders:
5         print(f"Printing Order #{order.id} - {order.customer.name}", "\n", "-" * 60)
6         for item in order.items.all():
7             print("Product: ", item.product.name)
8             print("Category: ", item.product.category.name)
9             print("Subtotal: ", item.subtotal, "\n")
```

45 queries (em 5 pedidos)

[45] SQL statements executed (Total time = 95.9ms, SQL time = 68.0ms):

1 FROM "shop_order" LIMIT 5

2 FROM "shop_customer" WHERE "shop_customer"."id" = 511 LIMIT 21

3 FROM "shop_orderitem" WHERE "shop_orderitem"."order_id" = 2885

4 FROM "shop_product" WHERE "shop_product"."id" = 963 LIMIT 21

5 FROM "shop_productcategory" WHERE "shop_productcategory"."id" = 24 LIMIT 21

6 FROM "shop_customer" WHERE "shop_customer"."id" = 412 LIMIT 21

Resolvendo:

```
1 @debug_queries()
2 def list_multiple_orders_and_items_good(limit=5):
3     prefetch_items = Prefetch(
4         "items", queryset=OrderItem.objects.select_related("product__category")
5     )
6     orders = (
7         Order.objects.select_related("customer")
8         .prefetch_related(prefetch_items)
9         .filter()[:limit]
10    )
11    for order in orders:
12        print(f"Printing Order #{order.id} - {order.customer.name}", "\n", "-" * 60)
13        for item in order.items.all(): -> Items já estão carregados
14            print("Product: ", item.product.name)
15            print("Category: ", item.product.category.name)
16            print("Subtotal: ", item.subtotal, "\n")
```

De 45 para 2 queries

91% mais rápido que o anterior

[2] SQL statements executed (Total time = 8.4ms, SQL time = 5.0ms):

```
1 FROM "shop_order" INNER JOIN "shop_customer" ON ("shop_order"."customer_id" =  
"shop_customer"."id") LIMIT 5
```

```
2 FROM "shop_orderitem" INNER JOIN "shop_product" ON ("shop_orderitem"."product_  
id" = "shop_product"."id") INNER JOIN "shop_productcategory" ON ("shop_product".  
"category_id" = "shop_productcategory"."id") WHERE "shop_orderitem"."order_id" I  
N (481, 3793, 56, 2070, 3439)
```



Select related OU prefetch related?

👉 Select Related:

- Utiliza um **JOIN** para trazer registros com relação direta

Por ex:

- **Order.customer**
- **Product.category**
- **Item.product**

👉 Prefetch Related:

- Utiliza uma segunda query para trazer registros relacionados (1 ou vários):
- Pode ser combinado com select related

Por ex:

- **order.items**

```
1 Order.objects.select_related("customer")
2 Product.objects.select_related("category")
3 OrderItem.objects.select_related("product__category")
```

```
1 Order.objects.prefetch_related("items")
```

Aprendizados com ORM



Ao escrever queries com ORM, sempre considerar as relações que serão utilizadas



SQL é nosso amigo e nos ajuda a identificar problemas



O ORM oferece vantagens sobre o SQL puro, pois não precisamos nos preocupar com a sintaxe do SQL no nosso código



Os mesmos problemas mostrados anteriormente podem acontecer no Django Admin. Cuidado com o método `__str__(self)`

Índices no banco de dados



O que são índices?



Como saber se preciso colocar um índice?



Analisando o Query Plan do **Postgres**



Verificando se o índice é realmente usado



Jorge

Chefe na empresa Kats

Chico, vi que você está mandando muito bem na otimização de queries, poderia acertar essa outra página também?

Otimizando além do ORM



Tarefa: Otimizar uma página lenta no website. A página lista o total vendido filtrando por email

```
1 @debug_queries(DebugTypes.FULL)
2 def list_total_sold_for_email_bad(email="bramirez@example.com"):
3     total_sold_for_email = Order.objects.filter(
4         customer__email__non_indexed=email
5     ).aggregate(total_sold=Sum("total"))
6
7     print(f"Total sold for {email}: ${total_sold_for_email['total_sold']}")
```

Sem problemas aparentes na query

```
In [15]: list_total_sold_for_email_bad()  
Total sold for bramirez@example.com: $1916.25
```

```
-----  
[1] SQL statements executed (Total time = 16.4ms, SQL time = 16.0ms):
```

```
1 SELECT SUM("shop_order"."total") AS "total_sold" FROM "shop_order" INNER JOIN  
  "shop_customer" ON ("shop_order"."customer_id" = "shop_customer"."id") WHERE "  
shop_customer"."email_non_indexed" = 'bramirez@example.com'
```



O que são índices?



Analizando o query plan

```
In [25]: print("\n", Order.objects.filter(customer__email_non_indexed="bramirez@example.com").explain(ANALYZE=True))
```

```
Nested Loop (cost=0.42..4536.75 rows=2 width=50) (actual time=1.011..14.368 rows=11 loops=1)
```

```
-> Seq Scan on shop_customer (cost=0.00..4516.01 rows=1 width=8) (actual time=0.952..13.958 rows=1 loops=1)
```

```
Filter: ((email_non_indexed)::text = 'bramirez@example.com'::text)
```

```
Rows Removed by Filter: 112000
```

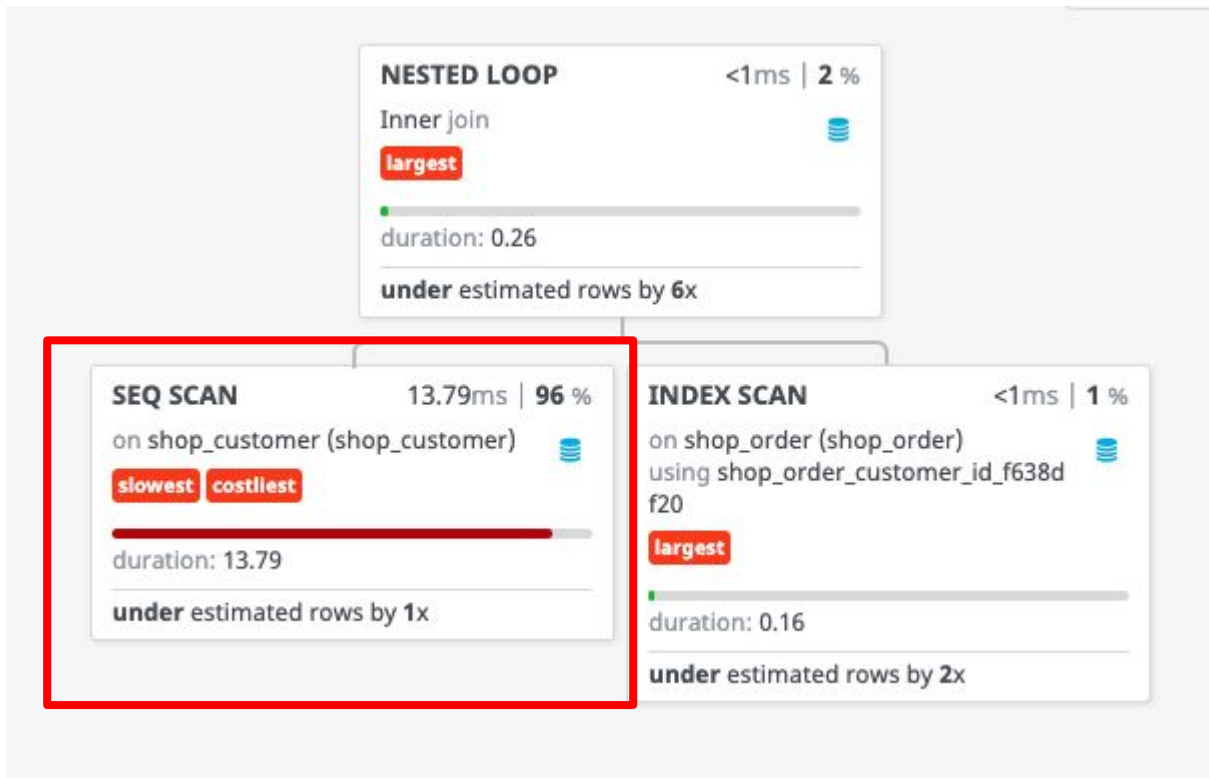
```
-> Index Scan using shop_order_customer_id_f638df20 on shop_order (cost=0.42..20.66 rows=7 width=50) (actual time=0.000..0.000 rows=7 loops=1)
```

```
Index Cond: (customer_id = shop_customer.id)
```

```
Planning Time: 0.220 ms
```

```
Execution Time: 14.521 ms
```

Analizando o query plan (visual)



From: <https://tatiyants.com/pev/#!/plans/new>

Colocando index

```
12 12 class Customer(BaseModel):
13 13     name = models.CharField(max_length=100)
14 14     birth_date = models.DateField()
15 -   email = models.EmailField(null=True, blank=True)
    15 +   email = models.EmailField(null=True, blank=True, db_index=True)
```

```
CREATE INDEX "shop_customer_email_d3fdf104" ON "shop_customer" ("email");
CREATE INDEX "shop_customer_email_d3fdf104_like" ON "shop_customer" ("email" varchar_pattern_ops);
```

Novo resultado

82% mais rápido que o anterior

```
In [29]: list_total_sold_for_email_good()  
Total sold for bramirez@example.com: $1916.25
```

```
-----  
[1] SQL statements executed (Total time = 2.8ms, SQL time = 2.0ms):
```

```
1 SELECT SUM("shop_order"."total") AS "total_sold" FROM "shop_order" INNER JOIN  
  "shop_customer" ON ("shop_order"."customer_id" = "shop_customer"."id") WHERE "  
shop_customer"."email" = 'bramirez@example.com'
```

```
-----
```


Novo query plan após indexar

```
In [30]: print("\n", Order.objects.filter(customer__email="bramirez@example.com").explain(ANALYZE=True))
```

```
Nested Loop (cost=0.84..29.17 rows=2 width=50) (actual time=0.073..0.441 rows=11 loops=1)
```

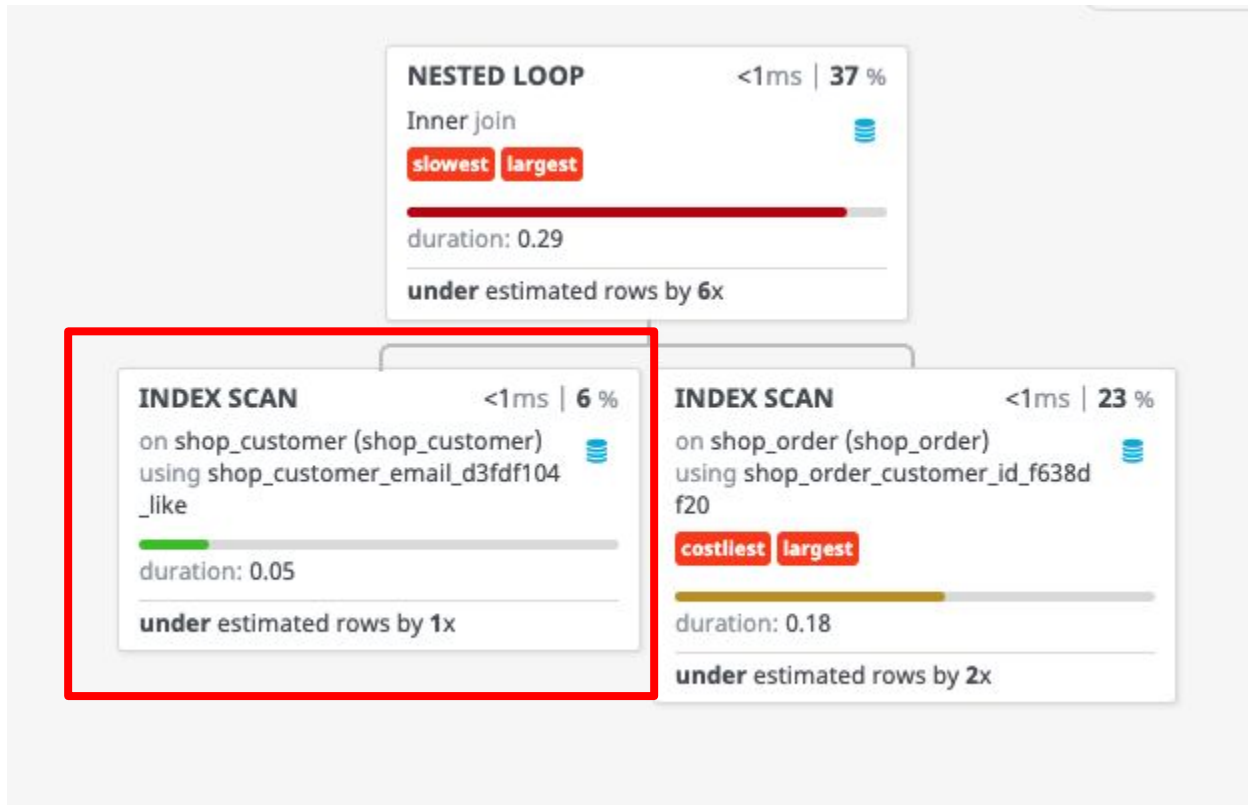
```
-> Index Scan using shop_customer_email_d3fdf104_like on shop_customer (cost=0.42..8.44 rows=1 width=8) (actual time=0.073..0.441 rows=11 loops=1)  
    Index Cond: ((email)::text = 'bramirez@example.com'::text)
```

```
-> Index Scan using shop_order_customer_id_f638df20 on shop_order (cost=0.42..20.66 rows=7 width=50) (actual time=0.073..0.441 rows=11 loops=1)  
    Index Cond: (customer_id = shop_customer.id)
```

```
Planning Time: 0.217 ms
```

```
Execution Time: 0.686 ms
```

Novo query plan (visual)



Trade offs índices

+ Considerar o tempo de inserção

🤔 O banco de dados pode não usar o índice

🏔 Volumetria importa

🕒 Espaço X tempo

Espaço index

| table_name | Total Size | Index Size | Actual Size |
|---------------|------------|------------|-------------|
| shop_customer | 29 MB | 4936 kB | 24 MB |
| (1 row) | | | |

Aumento de 31% no armazenamento para
contemplar o índice

| table_name | Total Size | Index Size | Actual Size |
|---------------|------------|------------|-------------|
| shop_customer | 38 MB | 13 MB | 24 MB |
| (1 row) | | | |

Resultados

| Operação | Problema | Solução | Melhoria |
|------------------------------------|--------------------------------|-----------------------------------|--|
| Listar pedidos | Queries extras | Select_related | -76% 13.8ms -> 3.2ms |
| Listar um pedido com itens | Queries extras | Select_related em ambas queries | -73% 24.2ms -> 6.4ms |
| Listar múltiplos pedidos com itens | Queries extras | Select_related + prefetch_related | -91% 95.9ms -> 8.4ms |
| Consulta de orders por email | Falta de índice no campo email | Indexar o email | -82% 16ms -> 2.8ms |

Managers - Organização

```
1 class OrderManager(models.Manager):
2     def get_queryset(self):
3         return super().get_queryset().select_related("customer")
4
5     def with_items(self):
6         OrderItem = apps.get_model("shop", "OrderItem")
7
8         prefetch_items = Prefetch(
9             "items", queryset=OrderItem.objects.select_related("product__category")
10        )
11        return self.get_queryset().prefetch_related(prefetch_items)
```

Managers - Organização

```
1 @debug_queries()
2 def list_multiple_orders_and_items_good_refactored(limit=5):
3     orders = Order.objects.with_items()[:limit]
4     for order in orders:
5         print(f"Printing Order #{order.id} - {order.customer.name}", "\n", "-" * 60)
6         for item in order.items.all():
7             print("Product: ", item.product.name)
8             print("Category: ", item.product.category.name)
9             print("Subtotal: ", item.subtotal, "\n")
```

Signals - Código + desacoplado

```
1 from django.db.models.signals import post_save
2 from django.dispatch import receiver
3 from shop.models import Order
4
5
6 @receiver(post_save, sender=Order)
7 def handle_new_order_created(sender, instance, created, **kwargs):
8     if created:
9         send_order_confirmation_email(instance)
10        update_accounting_software(instance)
11        do_something_cool(instance)
12        update_top_sales_person_rank(instance)
```


Extra: Queries mais complexas

```
1 @debug_queries(DebugTypes.FULL)
2 def list_product_top_sales(has_sales=True, limit=5):
3     produtos_with_sales = (
4         Product.objects.annotate(
5             has_sales=Exists(OrderItem.objects.filter(product_id=OuterRef("id"))),
6             total_sold=Sum("items_sold__subtotal"),
7         )
8         .filter(has_sales=has_sales)
9         .order_by("-total_sold")
10        .values_list("name", "total_sold")[:limit]
11    )
12
13    for name, total_sold in produtos_with_sales:
14        print("Product: ", name)
15        print("Total Sold: ", total_sold, "\n")
```

Extra: Quando as queries são executadas?

```
1 @debug_queries()
2 def list_queryset_evaluation():
3     queryset = Product.objects.all()
4     queryset = queryset.filter(id__in=[1, 2, 4, 5, 6, 7, 8, 9])
5     queryset = queryset.exclude(id__in=[2, 4])
6     queryset = queryset.annotate(extra_field=Value("Testing"))
7
8     for product in queryset:
9         print(product.name)
```

Quando as queries são executadas?

```
In [4]: list_queryset_evaluation()
```

```
Joan Reyes
```

```
David Miller
```

```
Ryan Little
```

```
Darren Suarez
```

```
Jasmine Salazar
```

```
Michele Blair
```

```
-----  
[1] SQL statements executed (Total time = 3.5ms, SQL time = 2.0ms):
```

```
1 FROM "shop_product" WHERE ("shop_product"."id" IN (1, 2, 4, 5, 6, 7, 8, 9) AND NOT ("shop_product"."id" IN (2, 4)))
```

Quando as queries são executadas?

Querysets são “Lazy”, ou seja, não executam a query no banco de dados antes dos dados serem necessários

Quando os dados são “necessários”?

- Ao se fazer uma iteração (for loop)
- Executar `len(queryset)`
- Slicing: `queryset[:10]`
- Listing: `list(queryset)`
- Calling `str()` and `repr()`
- Pickling/caching

Ferramentas auxiliares (em dev)



django-extensions:

- `python manage.py shell_plus --print-sql`
- `python manage.py dbshell`



django-debug-toolbar:

- Shows a debug toolbar on the web page



django-queryinspect

- Mostra estatísticas do SQL, ex:
- [SQL] 17 queries (4 duplicates), 34 ms SQL time, 243 ms total request time



Testes unitários

- `self.assertNumQueries(5)`



django-silk

- Profiling for Django

Ferramentas auxiliares (em prod)

Ferramentas de observabilidade e monitoramento em geral:

- Elastic APM
- New Relic
- Data Dog
- Honeycomb
- opentelemetry

Conclusão



O ORM não é lento, só pode estar mal configurado



Entender como os índices funcionam e como verificá-los nos ajuda a resolver problemas mais complexos




Existem muitas ferramentas disponíveis para auxiliar no debug e monitoramento de queries






Sempre testar o antes e depois de mudanças para garantir que elas realmente são efetivas.

Projeto de exemplo










<https://github.com/thiagoferreiraw/django-orm-optimization-talk>



 thiagoferreiraw / django-orm-optimization-talk Public

[Code](#) [Issues](#) [Pull requests](#) [Actions](#) [Projects](#) [Wiki](#) [Security](#) [Insights](#) [Settings](#)

 main  1 branch  0 tags

[Go to file](#) [Add file](#) [Code](#)

| | | |
|---|---|--|
|  | thiagoferreiraw docs: add query samples on readme | c78184c 19 minutes ago 23 commits |
|  | app | feat: added postgres database 6 hours ago |
|  | shop | feat: added product top sales 29 minutes ago |
|  | .gitignore | Initial commit 5 days ago |
|  | README.md | docs: add query samples on readme 19 minutes ago |
|  | docker-compose.yml | feat: added postgres database 6 hours ago |
|  | manage.py | feat: created django app 5 days ago |
|  | poetry.lock | feat: added postgres database 6 hours ago |
|  | pyproject.toml | feat: added postgres database 6 hours ago |

 README.md 

django-orm-optimization-talk

Muito Obrigado!

Bora pra discussão :)



Projeto de exemplo
dos slides



Github: @thiagoferreiraw

Twitter e Medium: @tferreiraw

Linkedin:

<https://www.linkedin.com/in/thiago-ferreira-380427a8/>

github.com/thiagoferreriraw/django-orm-optimization-talk